PSoC™ 4 high voltage (HV) precision analog (PA) MCU architecture

Reference manual

About this document

Scope and purpose

This document explains the architecture and functioning of the PSoC™ 4 high voltage (HV) precision analog (PA) MCU devices.

Intended audience

Engineers who need to become familiar with the hardware and firmware design principles of PSoC™ 4 HV PA MCU devices.

Table of contents

Table of contents

Table of contents

Section A: Overview

This section encompasses the following chapters:

1 Introduction

PSoC™ 4 HV Precision Analog is a fully-integrated programmable embedded system for battery monitoring and management. The system features an Arm® Cortex® M0+ processor and programmable and reconfigurable analog and digital blocks. The PSoC™ 4 HV PA devices have these characteristics:

- High-performance, 24 to 49.152-MHz Arm® Cortex® M0+ CPU with MPU and DMA controller
- Precision analog channel subsystem
- High-voltage subsystem (tolerant up to 42 V)
- Integrated LIN transceiver
- High-precision clock sources
- Configurable Timer/Counter/PWM block
- Configurable serial communication block with I^{2}C, SPI, and LIN slave operating modes
- Low-power operating modes: Sleep and Deep Sleep
- Functional Safety for ASIL-B as per ISO 26262
- Automotive Electronics Council (AEC) AEC-Q100 Qualified

This document describes each functional block of the PSoC™ HV PA devices in detail. This information will help designers to create system-level designs.

1.1 Top-level architecture

Figure 1-1 shows the major components of the PSoC™ 4 HV PA architecture.

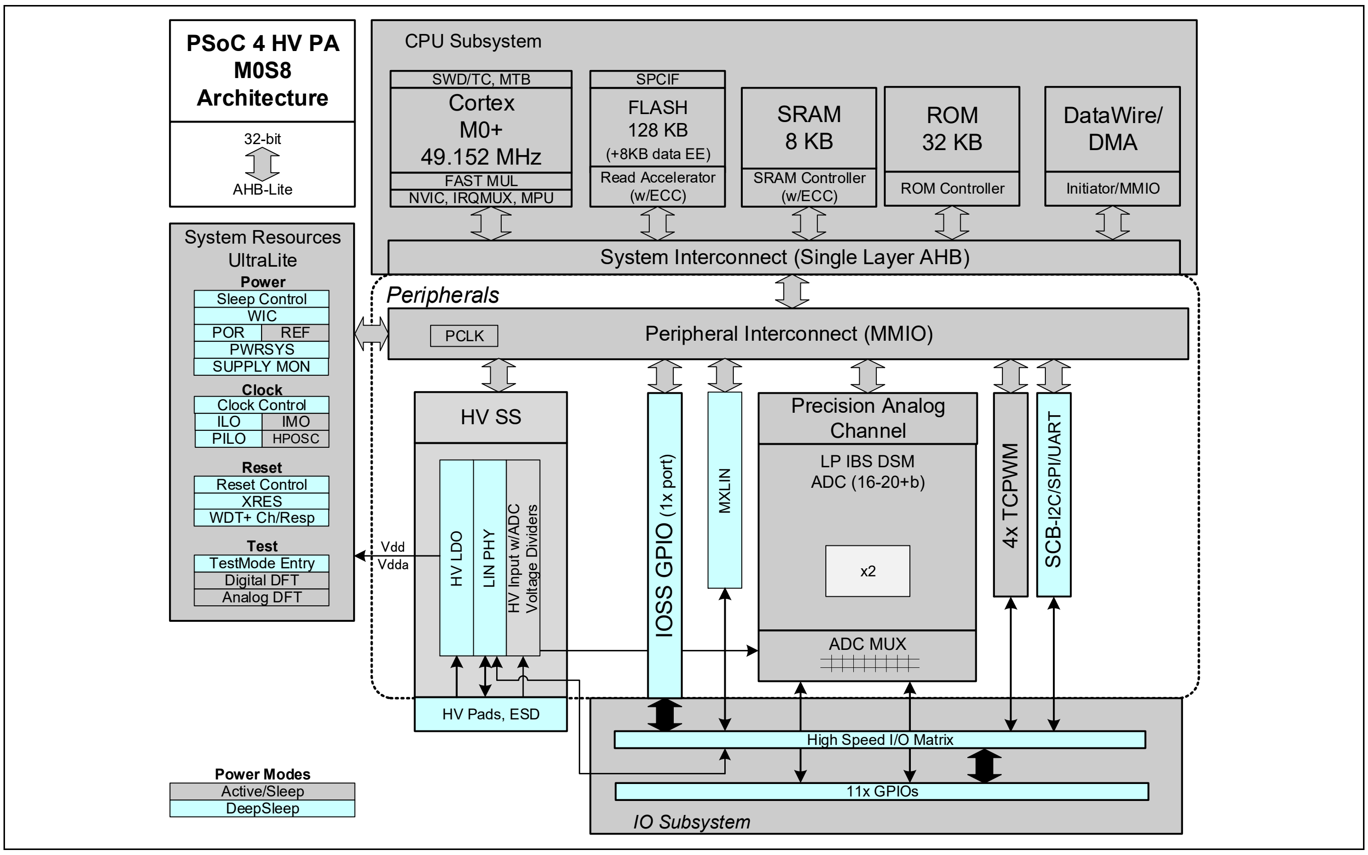


Figure 1-1. PSoC™ 4 HV PA block diagram

1.2 Features

The PSoC™ 4 HV PA device has these major components:

- 32-bit MCU Subsystem
  - 24 to 49.152-MHz Arm® Cortex® M0+ CPU with DMA
  - Up to 128KB of code flash with ECC
  - Up to 8KB of SRAM with ECC
  - Up to 8KB of data flash with ECC
- Precision Analog
  - Two precision ΔΣ ADCs (16 to 20+ bits)
  - Current channel with automatic gain
  - Voltage channel with HV input divider
  - Temperature and diagnostic channels
  - Digital filtering, accumulators, and threshold comparisons on all channels
- High-voltage subsystem
  - Operates directly off 12 V/24 V battery (tolerant up to 42 V)
  - Integrated LIN transceiver
  - ADC input voltage divider
- Functional Safety for ASIL-B
  - Development process of ISO 26262 for ASIL B
  - Memory Protection Unit (MPU)
  - Challenge-Response Watchdog Timer (CRWDT)
  - Detection of overvoltage and brownout events
  - Hardware error correction (SECDED ECC)
  - Analog diagnostics
- Timing and Pulse-Width Modulation
  - Four 16-bit timer/counter/PWM (TCPWM) blocks
  - Center-aligned, edge, and pseudo-random modes
  - Quadrature decoder
- Clock Sources
  - ±2% up to 49.152-MHz Internal Main Oscillator (IMO)
  - ±1% 2-MHz High-Precision Oscillator (HPOSC)
  - 40-kHz Internal Low-speed Oscillator (ILO)
  - ±5% or ±7% 32-kHz Precision Low-power Oscillator (PILO) based on the part number
- Communication
  - Serial communication blocks (SCBs) with reconfigurable I^{2}C, SPI, UART, or LIN Slave
  - Local Interconnect Network (LIN) block with LIN 2.2A
  - Up to 11 GPIOs

1.3 CPU system

1.3.1 Processor

The Cortex®-M0+ CPU in the PSoC™ 4 HV PA is part of the 32-bit MCU subsystem, which is optimized for low-power operation with extensive clock gating. Most instructions are 16 bits in length and execute a subset of the Thumb-2 instruction set. The Infineon implementation includes a hardware multiplier that provides a 32-bit result in one cycle.Programs can execute from SROM, SRAM, or FLASH memory.

The CPU also includes a debug interface, the serial wire debug (SWD) interface, which is a 2-wire form of JTAG; the debug configuration used for PSoC™ 4 HV PA has four break-point (address) comparators and two watchpoint (data) comparators.

1.3.2 Interrupt controller

The CPU subsystem includes a nested vectored interrupt controller (NVIC) block with 32 interrupt inputs and includes a Wakeup Interrupt Controller (WIC) in the system resources subsystem (SRSS), which can wake the processor up from the Deep Sleep mode allowing power to be switched off to the CPU when the chip is in the Deep Sleep mode. The Cortex®-M0+ CPU provides a Non-Maskable Interrupt (NMI) input, which is the highest priority exception other than reset.

1.3.3 Direct memory access

The DMA engine with eight channels is provided that can do 32-bit transfers and has chainable ping-pong descriptors. This DMA engine allows data transfer between memory, registers, and peripherals without CPU intervention. DMA transfers can occur while CPU is sleep. Descriptors identify the data source and destination along with other information.

1.4 Memory with ECC

Flash and SRAM include Error Correction Code (ECC) circuitry capable of correcting single-bit errors and detecting 2-bit errors. If a single bit error occurs, the data is corrected in-line, error information is stored, and an error flag is set, which can generate an interrupt. If a multi-bit error is detected, the error information is stored and either an interrupt or reset is generated.

1.4.1 Flash

The PSoC™ 4 HV PA has a flash module with separate controllers for code flash and data flash. The flash is with an accelerator, tightly coupled to the CPU to improve average access times from the flash block. The flash accelerator delivers 85% single-cycle SRAM access performance on average. Part of a flash module can be used to emulate EEPROM.

1.4.2 SRAM

Volatile static memory (SRAM) is used by the processor for storing variables and can program code can be written and executed in SRAM. SRAM memory is retained in all power modes (Active, Sleep, and Deep Sleep). At power-up, SRAM is uninitialized and should be written by application code before reading.

1.4.3 SROM

A supervisory read-only memory (ROM) contains boot and configuration routines that cannot be modified.

1.5 System resources

1.5.1 Power system

The power system includes regulators to generate appropriate voltages. The PSoC™ 4 HV PA operates at full performance from a single supply on V_{BAT} over a voltage range of 3.6 V to 28 V and remains functional up to 42 V. In addition to an active mode, the PSoC™ 4 HV PA has two low-power modes called Sleep and Deep Sleep. Transitions between the three power modes are managed by the power system in the SRSS.

The high-voltage regulator generates a 3.3-V supply from V_{BAT} for V_{DDD} and V_{DDA}. V_{DDA} powers analog circuits, while V_{DDD} provides power for I/Os (GPIOs) and the 1.8-V Core power regulators. There are different internal core regulators to support the various power modes. These include Active Digital regulator and Deep Sleep regulator.

1.5.2 Clock system

The PSoC™ 4 HV PA clock system is responsible for providing clocks to all subsystems that require clocks, for switching between different clock sources without glitching, and for synchronizing clocks operating on different frequency domains to prevent meta-stable conditions.

Four oscillators are implemented:

- A internal main oscillator (IMO) for CPU and peripheral clock generation, which is usually configured for 24-MHz to 48-MHz (in 4-MHz steps) and can support a “special” 49.152-MHz frequency
- A high-precision fixed frequency 2-MHz oscillator for precision timing (HPOSC)
- A low-power 40-kHz low-speed oscillator (ILO)
- A 32-kHz precision low-power oscillator (PILO) for wakeup timers and watchdog timers

There are also provisions for an external clock supplied by a GPIO pin. The ILO and PILO are permanently powered in all power modes.

1.5.3 GPIO

The PSoC™ 4 HV PA has eight GPIOs arranged in one port. The GPIOs have these features:

- Output drive modes include push-pull (strong or weak), open drain/source, high-z, and pull-up/-down
- Selectable CMOS and low-voltage LVTTL input buffer mode
- Edge-triggered interrupts on rising edge, falling edge, or on both the edges, on pin basis
- Individual control of input and output disables
- Hold mode for latching previous state (to retain I/O state in Deep Sleep)
- Selectable slew rates allowing dV/dt control to assist with noise control to improve EMI

1.5.4 Watchdog timers

The watchdog timer (WDT) is used to automatically reset the device in the event of an unexpected firmware execution path. It is also used as a wakeup source to periodically generate interrupts as a wakeup source in low-power modes.

A challenge-response watchdog (CRWDT) is available, which includes a window watchdog function, generating timeout events if the challenge-response watchdog is serviced too soon, too late, or with the wrong software key.

A lifetime counter is also available, which is used to calculate total time the battery is connected.

1.6 Digital system

1.6.1 Timer/Counter/PWM Block

The TCPWM in PSoC™ 4 HV PA implements 16-bit timer, counter, pulse width modulator (PWM), and quadrature decoder functionality. The PSoC™ 4 HV PA has four TCPWM blocks. The block can be used to measure the period and pulse width of an input signal (timer), count several events (counter), generate PWM signals, or decode quadrature signals.

1.6.2 Serial communication blocks

The serial communication blocks (SCB) supports four serial interface protocols: SPI, I^2C, UART, and LIN slave. Only one of the protocols is supported by an SCB at any given time. The PSoC™ 4 HV PA has one SCB.

This block supports the following features:

- Standard UART transmitter and receiver function
- LIN slave functionality with LIN v1.3 and LIN v2.1/2.2/2.2A specification compliance
- SPI master and slave functionality with Motorola, TI, and NSC protocols
- I^2C master and slave functionality
- EZ mode for SPI and I^2C, which allows for operation without CPU intervention

1.7 Analog system

1.7.1 Precision analog channel subsystem

The PSoC™ 4 HV PA precision analog channel subsystem (PACSS) is a high-performance data acquisition system consisting with two delta-sigma analog-to-digital converters (ADCs) and support circuitry. The two ADCs can quickly switch between input sources to create a third “virtual” ADC. The PACSS includes an analog input multiplexer, input buffer amplifiers, delta-sigma modulators, decimators, and digital signal processing channels. The PCASS also has a precision voltage reference, current references, and temperature sensors.

1.8 High-voltage subsystem

The PSoC™ 4 HV PA high-voltage subsystem includes the following functions:

- An AHB bus interface and control/status registers
- V_{BAT} to V_{DDD}/V_{DDA} HV regulator (3.6 V to 28 V input, 3.3-V nominal outputs)
- An input attenuator/voltage divider for VSENSE and VDIAG ADC inputs
- A LIN transceiver (physical interface or PHY)

1.8.1 HV regulator

The high-voltage regulator is always on, supplied by V_{BAT}, and provides V_{DD} and V_{DDA}. It supplies a nominal output voltage of 3.3 V but may drop as low as 2.7 V when V_{BAT} drops below 4 V.

1.8.2 LIN transceiver

The LIN transceiver meets the requirements of LIN standard 2.2A and is downward compatible with the LIN 2.0. Data rates of 10 kB/sec and 20 kB/sec are supported. A non-LIN fast slew rates is available providing 100 kB/sec data rates for fast downloads for factory and field flash program updates using the LIN pin.

1.9 Program and debug

PSoC™ 4 HV PA devices include extensive support for programming, testing, debugging, and tracing both hardware and firmware. The Arm® SWD interface supports all programming and debug features of the device.

1.10 Functional overview

Table 1-1. PSoC™ 4 HV PA device summary

Feature	PSoC™ 4 HV PA
CPU	
Core	32-bit Cortex® M0+ CPU
Maximum frequency	24 to 49.152 MHz
DMA	8
Interrupt	32
SysTick timer	Available
Fault subsystem	Available
Memory	
Code flash with ECC	Up to 128KB
Data flash with ECC	Up to 8KB
SRAM with ECC	Up to 8KB
Precision analog	
Precision ΔΣ ADCs (16–20+ bits)	2
Current channel with automatic gain	2
Voltage channel with HV input divider	2
Temperature and diagnostic channels	Temperature:1, Diagnostic:1
Temperature sensing	Internal/External
Digital channel data path	4ch (2 with FIR, 2 without FIR) Digital filtering, accumulators, threshold comparisons on all channels
High-voltage subsystem	
Operation voltage	3.6 V to 28 V Remains functional up to 42 V
High-voltage regulator	3.3 V nominal output
LIN transceiver	1
ADC input voltage divider	2
Digital system	
16-bit timer/counter/pulse-width modulator (TCPWM)	4
Serial communication block (I^2C, SPI, UART or LIN)	1
Independent LIN block	1
Clock sources	
Internal main oscillator (IMO)	±2% up to 49.152 MHz
High-precision oscillator (HPOSC)	±1% 2 MHz

Table 1-1. PSoC™ 4 HV PA device summary (continued)

Feature	**PSoC™ 4 HV PA**
Precision internal low-power oscillator (PILO)	±5% or ±7% 32 kHz based on the part number
Internal low-speed oscillator (ILO)	40 kHz
Functional safety for ASIL-B	
Memory protection unit (MPU)	Available
Basic watchdog timer (WDT)	1
Challenge-response watchdog timer (CRWDT)	1
Lifetime counter	1
Power supply monitoring	Overvoltage/Brownout
Hardware error correction	All safety-critical memories
Analog diagnostics	Backup reference voltage, redundancy in voltage, current and temperature measurement paths
System	
GPIOs	11
Debug interface	SWD
Package	32-pin QFN with wettable flanks (6 × 6 mm)
Operating temperature	–40°C to +125°C

2 Getting started

2.1 Support

Free support for PSoC™ 4 HV Precision Analog products is available online at www.infineon.com. Resources include training seminars, discussion forums, application notes, PSoC™ consultants, CRM technical support email, knowledge base, and application support engineers.

For application assistance, visit www.infineon.com/support/.

2.2 Product Upgrades

Infineon provides scheduled upgrades and version enhancements for the sample driver library (SDL) (which can be used only to evaluate the microcontroller and not for production usage) and PSoC™ Programmer free of charge. Upgrades are available at www.infineon.com. Critical updates to system documentation are also provided in the Documentation section.

2.3 Development Kits

The CYHVPA-128K-32-001 kit enables customers to evaluate and design with the PSoC™ 4 HV PA series of devices. The CYHVPA-128K-32-001 kit is Multi-functional board corresponding to all features. It includes peripherals that evaluate the key features of the PSoC™ 4 HV PA series of devices.

To request the kit, contact your local sales representative.

2.4 Application Notes

See the application note *AN230264 - Getting Started with PSoC™ 4 HV PA* for additional information on the PSoC™ 4 HV PA device capabilities and to quickly create a simple application using development kits.

3 Document construction

This document includes the following sections:

- **"Overview"** on page 11
- **"System Resources Subsystem (SRSS)"** on page 92
- **"I/O System"** on page 140
- **"Digital System"** on page 151
- **"Analog System"** on page 275
- **"High-Voltage System"** on page 344
- **"Program and Debug"** on page 360

3.1 Major sections

For ease of use, information is organized into sections and chapters that are divided according to device functionality.

- Section – Presents the top-level architecture, how to get started, and conventions and overview information of the product.
- Chapter – Presents the chapters specific to an individual aspect of the section topic. These are the detailed implementation and use information for some aspect of the integrated circuit.
- Glossary – Defines the specialized terminology used in this technical reference manual (TRM). Glossary terms are presented in bold, italic font throughout.
- Registers Technical Reference Manual – Supplies all device register details summarized in the technical reference manual. This is an additional document.

3.2 Documentation conventions

This document uses only four distinguishing font types, besides those found in the headings.

- The first is the use of *italics* when referencing a document title or file name.
- The second is the use of ***bold italics*** when referencing a term described in the Glossary of this document.
- The third is the use of Times New Roman font, distinguishing equation examples.
- The fourth is the use of `Courier New` font, distinguishing code examples.

3.2.1 Register conventions

Register conventions are detailed in the *PSoC™ 4 HV Precision Analog Registers TRM*.

3.2.2 Numeric naming

Hexadecimal numbers are represented with all letters in uppercase with an appended lowercase 'h' (for example, '14h' or '3Ah') and *hexadecimal* numbers may also be represented by a '0x' prefix, the *C* coding convention. Binary numbers have an appended lowercase 'b' (for example, 01010100b' or '01000011b'). Numbers not indicated by an 'h' or 'b' are *decimal*.

3.2.3 Units of measure

This table lists acronyms that might be used in this document.

Table 3-1. Units of measure

Abbreviation	Unit of measure
bps	bits per second
°C	degrees Celsius
dB	decibels
fF	femtofarads
Hz	Hertz
k	kilo, 1000
K	kilo, 2^10
KB	1024 bytes, or approximately one thousand bytes
Kbit	1024 bits
kHz	kilohertz (32.000)
kΩ	kilohms
MHz	megahertz
MΩ	megaohms
µA	microamperes
µF	microfarads
µs	microseconds
µV	microvolts
µVrms	microvolts root-mean-square
mA	milliamperes
ms	milliseconds
mV	millivolts
nA	nanoamperes
ns	nanoseconds
nV	nanovolts
W	ohms
pF	picofarads
pp	peak-to-peak
ppm	parts per million
SPS	samples per second
s	sigma: one standard deviation
V	volts

3.2.4 Acronyms

This table lists the acronyms used in this document.

Table 3-2. Acronyms

Acronym	Definition
ABUS	analog output bus
AC	alternating current
A/D	analog digital converter
ADC	analog-to-digital converter
AES	advanced encryption standard
AHB	AMBA (advanced microcontroller bus architecture) high-performance bus, an Arm® data transfer bus
API	application programming interface
APOR	analog power-on reset
AXI	advanced extensible interface
BC	broadcast clock
BDR	boot description record
BOD	brownout detect
BOM	bill of materials
BR	bit rate
BRA	bus request acknowledge
BRQ	bus request
CAN	controller area network
CI	carry in
CMP	compare
CO	carry out
COM	LCD common signal
CPU	central processing unit
CPUSS	CPU sub-system
CR	CR oscillator
CRC	cyclic redundancy check
CSV	clock supervisor
CT	continuous time
CTB	continuous time block
CTBm	continuous time block mini
DAC	digital-to-analog converter
DAP	debug access port
DC	direct current
DED	dual error detection
DI	digital or data input

Table 3-2. Acronyms (continued)

Acronym	Definition
DMA	direct memory access
DMAC	DMA controller
DMIPS	Dhrystone million instructions per second
DO	digital or data output
DSI	digital signal interface
DSM	Deep Sleep mode
DW	data wire
ECC	error correction code (safety)
ECO	external crystal oscillator
EEPROM	electrically erasable programmable read only memory
EMIF	external memory interface
EXT-IRC	external interrupt controller
FB	feedback
FIFO	first in first out
FIQ	fast interrupt request
FPU	floating point unit
FRT	free run timer
FSR	full scale range
GPIO	general purpose I/O
HCI	host-controller interface
HFCLK	high-frequency clock
HPM	high-performance matrix
HPMC	high performance motor control
HSIOM	high-speed I/O matrix
HVSS	high-voltage subsystem
HW-WDT	hardware watchdog timer
I^2C	inter-integrated circuit
ICU	input capture unit
IDE	integrated development environment
ILO	internal low-speed oscillator
IPC	inter-processor communication
IRC	interrupt controller
IRQ	interrupt request
ITO	indium tin oxide
IMO	internal main oscillator
INL	integral nonlinearity
I/O	input/output
IOSS	I/O sub-system

Table 3-2. **Acronyms** (continued)

Acronym	Definition
IOR	I/O read
IOW	I/O write
IRES	initial power on reset
IRA	interrupt request acknowledge
IRQ	interrupt request
ISR	interrupt service routine
IVR	interrupt vector read
JTAG	joint test action group
LCD	liquid crystal display
LFCLK	low-frequency clock
LIN	local interconnect network
LLPP	low-latency peripheral port
LIN	local interconnect network
LPCOMP	low-power comparator
LRb	last received bit
LRB	last received byte
LSb	least significant bit
LSB	least significant byte
LUT	lookup table
LVD	low-voltage detector
MCU	microcontroller unit
MISO	master-in-slave-out
MMIO	memory mapped input/output
MOSI	master-out-slave-in
MPU	memory protection unit
MSb	most significant bit
MSB	most significant byte
MSP	main stack pointer
NF	noise filter
NMI	non-maskable interrupt
NVIC	nested vectored interrupt controller
OCU	output compare unit
OSC	Oscillator
PACSS	precision analog channel sub-system
PASS	programmable analog sub-system
PC	program counter
PCB	printed circuit board
PCH	program counter high

Table 3-2. Acronyms (continued)

Acronym	Definition
PCL	program counter low
PD	power down
PGA	programmable gain amplifier
PLL	phase locked loop
PHY	physical layer
PM	power management
PMA	PSoC™ memory arbiter
POR	power-on reset
PPC	port pin configuration
PPU	peripheral protection unit
PPOR	precision power-on reset
PRS	pseudo random sequence
PSoC™	Programmable System-on-Chip
PSP	process stack pointer
PSRR	power supply rejection ratio
PSS	power saving state
PSSDC	power system sleep duty cycle
PWM	pulse width modulator
RAM	random-access memory
RETI	return from interrupt
RF	radio frequency
RIC	resource input configuration
ROM	read only memory
RMS	root mean square
RMW	read modify write
RTC	real-time clock
RW	read/write
SAR	successive approximation register
SEG	LCD segment signal
SC	switched capacitor
SCB	serial communication block
SCT	source clock timer
SEC	single-error correction
SHA	secure hash algorithm
SHE	secure hardware extension
SIE	serial interface engine
SIO	special I/O
SE0	single-ended zero

Table 3-2. Acronyms (continued)

Acronym	Definition
SEooC	safety element out of context
SMC	stepping motor control
SNR	signal-to-noise ratio
SOC	state of charge
SOH	state of health
SOF	start of frame
SOI	start of instruction
SP	stack pointer
SPD	sequential phase detector
SPI	serial peripheral interconnect
SPIM	serial peripheral interconnect master
SPIS	serial peripheral interconnect slave
SRAM	static random-access memory
SROM	supervisory read only memory
SRSS	system resource sub-system
SSADC	single slope ADC
SSC	supervisory system call
SSCG	spectrum spread clock generator
SYSCLK	system clock
SWD	single wire debug
TC	terminal count
TCPWM	timer, counter, PWM
TD	transaction descriptors
TIA	trans-impedance amplifier
TRNG	true random number generator
UART	universal asynchronous receiver/transmitter
UDB	universal digital block
USB	universal serial bus
USBIO	USB I/O
VIC	vectored interrupt controller
VTOR	vector table offset register
WCO	watch crystal oscillator
WDT	watchdog timer
WDR	watchdog reset
XRES	external reset
XRES_N	external reset, active low
ZPD	zero point detection

4 CPU subsystem

This section encompasses the following chapters:

- **"Cortex®-M0+ CPU"** on page 28
- **"DMA controller modes"** on page 35
- **"Interrupts"** on page 54
- **"Flash Memory"** on page 67
- **"SRAM"** on page 79
- **"Fault subsystem"** on page 87

Top-level architecture

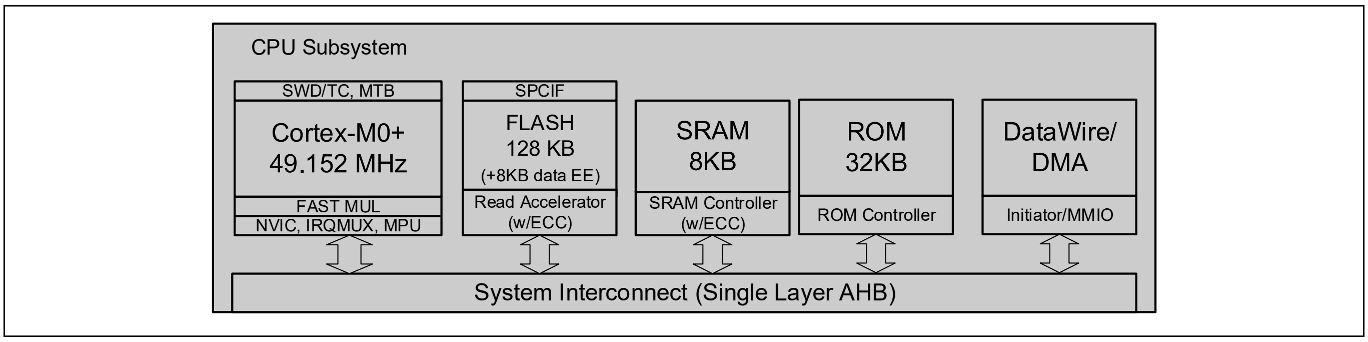


Figure 4-1. CPU System block diagram

5 Cortex®-M0+ CPU

The PSoC™ 4 HV PA Arm® Cortex®-M0+ core is a 32-bit CPU optimized for low-power operation. It has an efficient two-stage pipeline, a fixed 4-GB memory map, and supports the ARMv6-M Thumb instruction set. The Cortex®-M0+ also features a single-cycle 32-bit multiply instruction and low-latency interrupt handling. Other subsystems tightly linked to the CPU core include a nested vectored interrupt controller (NVIC), a SYSTICK timer, and debug.

This section gives an overview of the Cortex®-M0+ processor. For more details, see the Arm® Cortex®-M0+ user guide or technical reference manual, both available at www.arm.com.

5.1 Features

The PSoC™ 4 HV PA Cortex®-M0+ has the following features:

- Easy to use, program, and debug, ensuring easier migration from 8- and 16-bit processors
- Operates at up to 0.9 DMIPS/MHz; this helps to increase execution speed or reduce power
- Supports the Thumb instruction set for improved code density, ensuring efficient use of memory
- NVIC unit to support interrupts and exceptions for rapid and deterministic interrupt response
- Implements design time configurable Memory Protection Unit (MPU)
- Supports unprivileged and privileged mode execution
- Supports optional Vector Table Offset Register (VTOR)
- Extensive debug support including:
  - SWD port
  - Breakpoints
  - Watchpoints

5.2 Block diagram

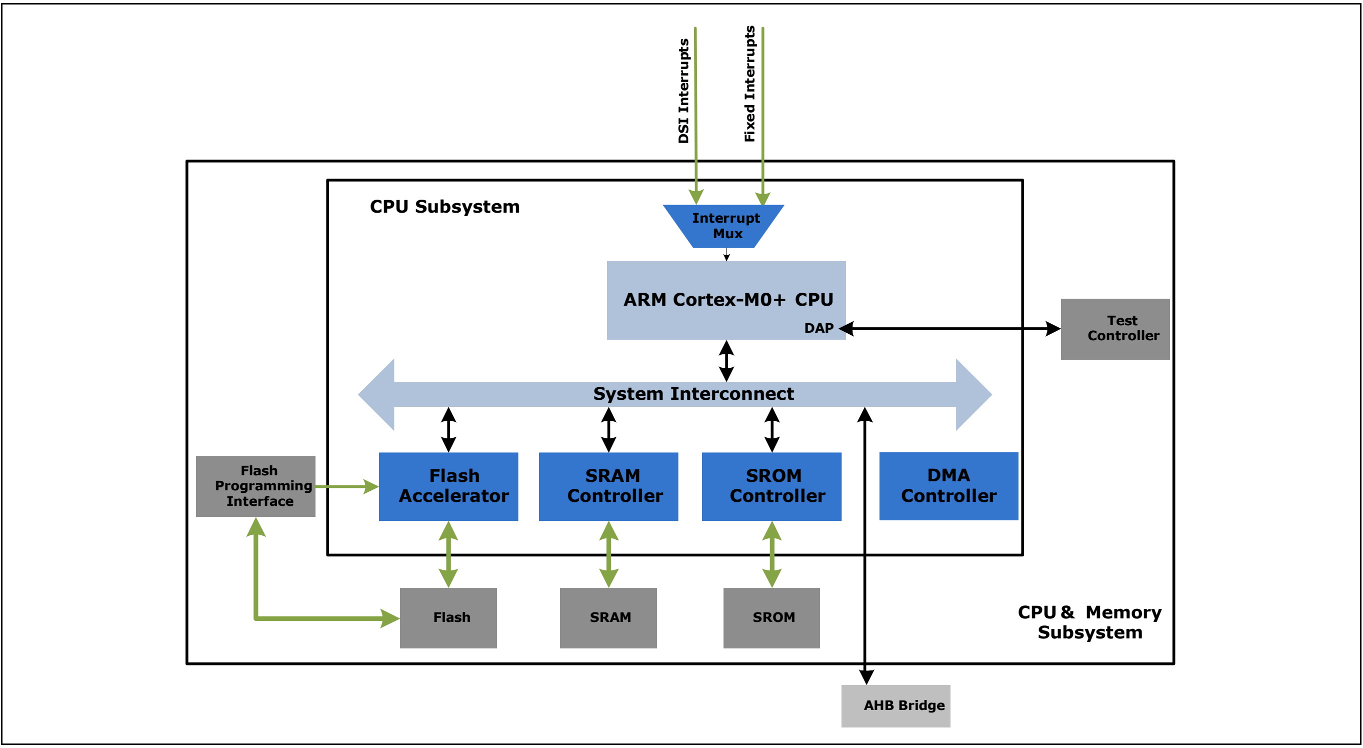


Figure 5-1. CPU subsystem block diagram

5.3 How it works

The Cortex®-M0+ is a 32-bit processor with a 32-bit data path, 32-bit registers, and a 32-bit memory interface. It supports most 16-bit instructions in the Thumb instruction set and some 32-bit instructions in the Thumb-2 instruction set.

The processor supports two operating modes (see **"Operating modes"** on page 31). It has a single-cycle 32-bit multiplication instruction.

5.4 Address map

The Arm® Cortex®-M0+ has a fixed address map allowing access to memory and peripherals using simple memory access instructions. The 32-bit (4 GB) address space is divided into the regions shown in **Table 5-1**. Note that code can be executed from the code and SRAM regions.

Table 5-1. Cortex®-M0+ address map

Address range	Name	Use
0x00000000–0x0001FFFF	Code FLASH	128KB Program code region. You can also place data here. Includes the exception vector table, which starts at address 0.
0x0FFFE000–0x0FFFE3FF	SFLASH	1KB Supervisory Flash Area (Trim & Wounding Info)
0x0FFFE400–0x0FFFE7FF	User SFLASH	1KB User Supervisory Flash Area (User data)
0x10000000–0x10007FFF	ROM	32KB ROM (Boot & Test)
0x1F000000–0x1F001FFF	Data FLASH	8KB Data region. A common usage is implementing a load balanced EEPROM for storing data.
0x20000000–0x20001FFF	SRAM	8KB System RAM (Data region)
0x40000000–0x5FFFFFFF	Peripheral	All peripheral registers. You cannot execute code from this region.
0x60000000–0xDFFFFFFF		Not used.
0xE0000000–0xE00FFFFF	PPB	Peripheral registers within the CPU core.
0xE0100000–0xFFFFFFFF	Device	PSoC™ 4 HV PA implementation-specific.

5.5 Registers

The Cortex®-M0+ has sixteen 32-bit registers, as **Table 5-2** shows:

- R0 to R12 – General-purpose registers. R0 to R7 can be accessed by all instructions; the other registers can be accessed by a subset of the instructions.
- R13 – Stack pointer (SP). There are two stack pointers, with only one available at a time. In thread mode, the CONTROL register indicates the stack pointer to use, Main Stack Pointer (MSP) or Process Stack Pointer (PSP).
- R14 – Link register. Stores the return program counter during function calls.
- R15 – Program counter. This register can be written to control program flow.

Table 5-2. Cortex®-M0+ Registers

Name	Type[a)]	Reset value	Description
R0–R12	RW	Undefined	R0–R12 are 32-bit general-purpose registers for data operations.
MSP (R13) PSP (R13)	RW	[0x00000000]	The stack pointer (SP) is register R13. In thread mode, bit[1] of the CONTROL register indicates which stack pointer to use: 0 = Main stack pointer (MSP). This is the reset value. 1 = Process stack pointer (PSP). On reset, the processor loads the MSP with the value from address 0x00000000.
LR (R14)	RW	Undefined	The link register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions.
PC (R15)	RW	[0x00000004]	The program counter (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value from address 0x00000004. Bit[0] of the value is loaded into the EPSR T-bit at reset and must be 1.
PSR	RW	Undefined	The program status register (PSR) combines: Application Program Status Register (APSR). Execution Program Status Register (EPSR). Interrupt Program Status Register (IPSR).
APSR	RW	Undefined	The APSR contains the current state of the condition flags from previous instruction executions.
EPSR	RO	[0x00000004].0	On reset, EPSR is loaded with the value bit[0] of the register [0x00000004].
IPSR	RO	0	The IPSR contains the exception number of the current ISR.
PRIMASK	RW	0	The PRIMASK register prevents activation of all exceptions with configurable priority.
CONTROL	RW	0	The CONTROL register controls the stack used when the processor is in thread mode.

a) Describes access type during program execution in thread mode and handler mode. Debug access can differ.

Table 5-3 shows how the PSR bits are assigned.

Table 5-3. Cortex®-M0+ PSR bit assignments

Bit	PSR Register	Name	Usage
31	APSR	N	Negative flag
30	APSR	Z	Zero flag
29	APSR	C	Carry or borrow flag
28	APSR	V	Overflow flag
27–25	–	–	Reserved
24	EPSR	T	Thumb state bit. Must always be 1. Attempting to execute instructions when the T bit is 0 results in a HardFault exception.
23–6	–	–	Reserved
5–0	IPSR	N/A	Exception number of current ISR: 0 = thread mode 1 = reserved 2 = NMI 3 = HardFault 4 – 10 = reserved 11 = SVCall 12, 13 = reserved 14 = PendSV 15 = SysTick 16 = IRQ0 ... 35 = IRQ19

Use the MSR or CPS instruction to set or clear bit 0 of the PRIMASK register. If the bit is 0, exceptions are enabled. If the bit is 1, all exceptions with configurable priority, that is, all exceptions except HardFault, NMI, and Reset, are disabled. See the Interrupts chapter on page 54 for a list of exceptions.

5.6 Operating modes

The Cortex®-M0+ processor supports two operating modes:

- Thread Mode – used by all normal applications. In this mode, the MSP or PSP can be used. The CONTROL register bit 1 determines which stack pointer is used:
  - 0 = MSP is the current stack pointer
  - 1 = PSP is the current stack pointer
- Handler Mode – used to execute exception handlers. The MSP is always used.

In thread mode, use the MSR instruction to set the stack pointer bit in the CONTROL register. When changing the stack pointer, use an ISB instruction immediately after the MSR instruction. This action ensures that instructions after the ISB execute using the new stack pointer.

In handler mode, explicit writes to the CONTROL register are ignored, because the MSP is always used. The exception entry and return mechanisms automatically update the CONTROL register.

5.7 Instruction set

The Cortex®-M0+ implements a version of the Thumb instruction set, as **Table 5-4** shows. For details, see the Cortex®-M0+ Generic User Guide.

An instruction operand can be an Arm® register, a constant, or another instruction-specific parameter. Instructions act on the operands and often store the result in a destination register. Many instructions are unable to use, or have restrictions on using, the PC or SP for the operands or destination register.

Table 5-4. Thumb instruction set

Mnemonic	Brief description
ADCS	Add with carry
ADD{S}[a)]	Add
ADR	PC-relative address to register
ANDS	Bit wise AND
ASRS	Arithmetic shift right
B{cc}	Branch {conditionally}
BICS	Bit clear
BKPT	Breakpoint
BL	Branch with link
BLX	Branch indirect with link
BX	Branch indirect
CMN	Compare negative
CMP	Compare
CPSID	Change processor state, disable interrupts
CPSIE	Change processor state, enable interrupts
DMB	Data memory barrier
DSB	Data synchronization barrier
EORS	Exclusive OR
ISB	Instruction synchronization barrier
LDM	Load multiple registers, increment after
LDR	Load register from PC-relative address
LDRB	Load register with word
LDRH	Load register with half-word
LDRSB	Load register with signed byte
LDRSH	Load register with signed half-word
LSLS	Logical shift left
LSRS	Logical shift right
MOV{S}[a]	Move
MRS	Move to general register from special register
MSR	Move to special register from general register
MULS	Multiply, 32-bit result

Table 5-4. Thumb instruction set (continued)

Mnemonic	Brief description
MVNS	Bit wise NOT
NOP	No operation
ORRS	Logical OR
POP	Pop registers from stack
PUSH	Push registers onto stack
REV	Byte-reverse word
REV16	Byte-reverse packed half-words
REVSH	Byte-reverse signed half-word
RORS	Rotate right
RSBS	Reverse subtract
SBCS	Subtract with carry
SEV	Send event
STM	Store multiple registers, increment after
STR	Store register as word
STRB	Store register as byte
STRH	Store register as half-word
SUB{S}[a]	Subtract
SVC	Supervisor call
SXTB	Sign extend byte
SXTH	Sign extend half-word
TST	Logical AND-based test
UXTB	Zero extend a byte
UXTH	Zero extend a half-word
WFE	Wait for event
WFI	Wait for interrupt

a) The ‘S’ qualifier causes the ADD, SUB, or MOV instructions to update APSR condition flags.

5.7.1 Address alignment

An aligned access is an operation where a word-aligned address is used for a word or multiple word access, or where a half-word-aligned address is used for a half-word access. Byte accesses are always aligned.

No support is provided for unaligned accesses on the Cortex®-M0+ processor. Any attempt to perform an unaligned memory access operation results in a HardFault exception.

5.7.2 Memory Endianness

The Cortex®-M0+ uses the little-endian format, where the least-significant byte of a word is stored at the lowest address and the most significant byte is stored at the highest address.

5.8 Systick timer

The Systick timer is integrated with the NVIC and generates the SYSTICK interrupt. This interrupt can be used for task management in a real-time system. The timer has a reload register with 24 bits available to use as a countdown value. The Systick timer uses either the Cortex®-M0+ internal clock or the low-frequency clock (LF_CLK) as the source.

5.9 Debug

PSoC™ 4 HV PA contains a debug interface based on SWD; it features four breakpoint (address) comparators and two watchpoint (data) comparators.

6 DMA controller modes

The DMA controller provides DataWire (DW) and Direct Memory Access (DMA) functionality. The DMA controller has the following features:

- Supports eight DMA channels
- Four levels of priority for each channel
- Byte (8 bits), half-word (16 bits), and word (32 bits) transfers
- Three modes of operation supported for each channel
- Configurable interrupt generation
- Output trigger on completion of transfer
- Transfer sizes up to 65,536 data elements

The DMA controller supports three operation modes. These operational modes are different in how the DMA controller operates on a single trigger signal. These operating modes allow the user to implement different operation scenarios for the DMA. The operation modes are

- Mode 0: Single data element per trigger
- Mode 1: All data elements per trigger
- Mode 2: All data elements per trigger and automatically trigger chained descriptor

The data transfer specifics, such as source and destination address locations and the size of the transfer, are specified by a descriptor structure. Each channel has an independent descriptor structure.

The DMA controller provides Active/Sleep functionality and is not available in the Deep Sleep power mode.

6.1 Block diagram description

The DMA transfers data to and from memory, peripherals, and registers. These transfers occur independent of the CPU. The DMA can transfer up to 65,536 data elements in one transfer. These data elements can be 8-bit, 16-bit, or 32-bit wide. The DMA starts each transaction through an external trigger that can come from a DMA channel (including itself), another DMA channel, a peripheral, or the CPU. The DMA is best used to offload data transfer tasks from the CPU.

Figure 6-1 gives an overview of the DMA controller at a block level.

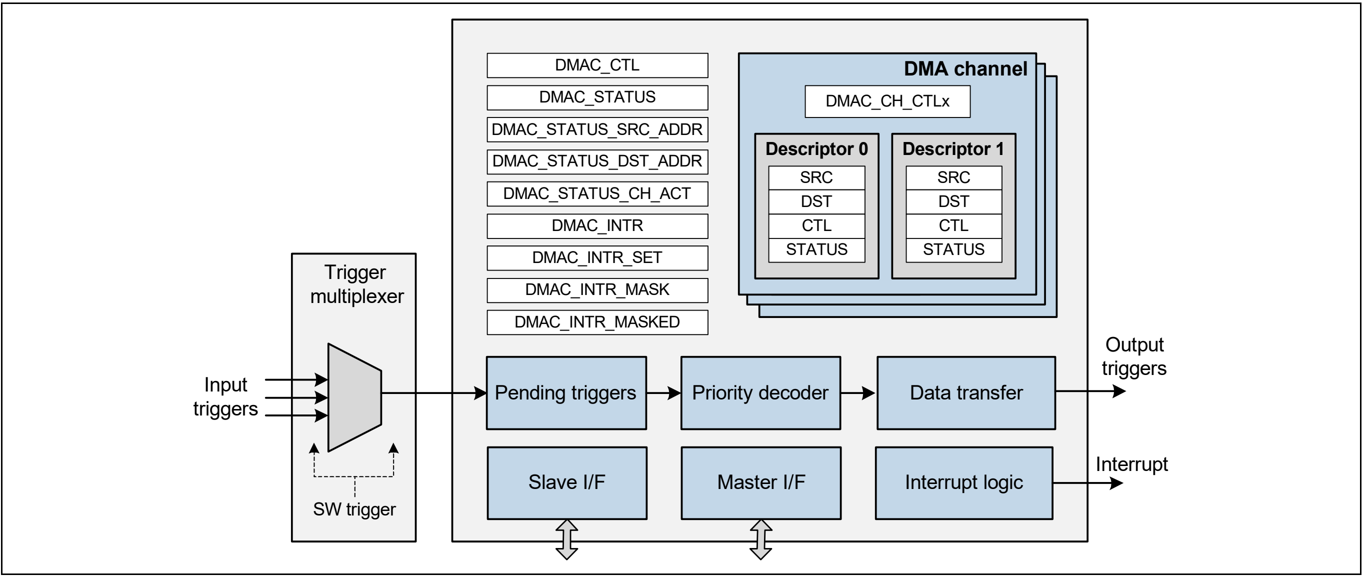


Figure 6-1. DMA Controller block diagram

Every DMA channel has two descriptors, which are responsible for configuring parameters specific to the transfer, such as source address, destination address, and data width. The transfer initiation in the DMA channel is on a trigger event. The trigger signals can come from different peripherals in the device, including the DMA itself.

The DMA controller has two bus interfaces, the master interface and the slave interface. Master I/F is an AHB-Lite bus master, which allows the DMA controller to initiate AHB-Lite data transfers to the source and destination locations. The DMA is the bus master in the master interface. This is the interface through which all DMA transfers are accomplished.

The DMA configuration registers and descriptors are accessed and reconfigured through the slave interface. Slave I/F is an AHB-Lite bus slave, which allows the PSoC™ main CPU to access the DMA controller's control/status registers and to access the descriptor structure. CPU is generally the master for this bus.

The receipt of a trigger activates a state machine in the DMA controller that goes through a trigger prioritization and processing and then initiates a data transfer according to the descriptor setting. When a transfer is complete, an output trigger is generated, which can be used as trigger condition or event for starting another function.

The DMA controller also has an interrupt logic block. Only one interrupt line is available from the DMA controller to interrupt the CPU. Individual DMA descriptors can be configured so that they activate this interrupt line on completion of the transfer.

6.2 Trigger sources and multiplexing

Every DMA channel has an input and output trigger associated with it. The input trigger can come from any peripheral, CPU, or a DMA channel itself. The input trigger is used to trigger a DMA transfer, as defined by the **"Transfer modes"** on page 44. A 'logic high', on the trigger input will trigger the DMA channel. The minimum width of this 'logic high' is two system clock cycles. The deactivation setting configures the nature of trigger deactivation.

The output trigger signals the completion of a transfer. This signal can be used as a trigger to a DMA channel or as a digital signal to the digital interconnect. The trigger input can come from different sources and is routed through a **"Trigger multiplexer"** on page 37.

6.3 Trigger multiplexer

The DMA channels can have trigger inputs from different peripheral sources in the PSoC™. This is routed to the individual DMA channel trigger inputs through the trigger multiplexer.

In the DMA trigger, multiplexers are organized in trigger groups. Each trigger group is composed of multiple multiplexers feeding into the individual DMA channel trigger inputs.

The PSoC™ 4 HV PA device implements a single trigger group (Trigger group 0), which provides trigger inputs to the DMA. The trigger input options can come from TCPWM, SCB, GPIO, DMA and Precision Analog Channel Subsystem (PACSS). **Figure 6-2** shows the PSoC™ 4 HV PA trigger multiplexer implementation.

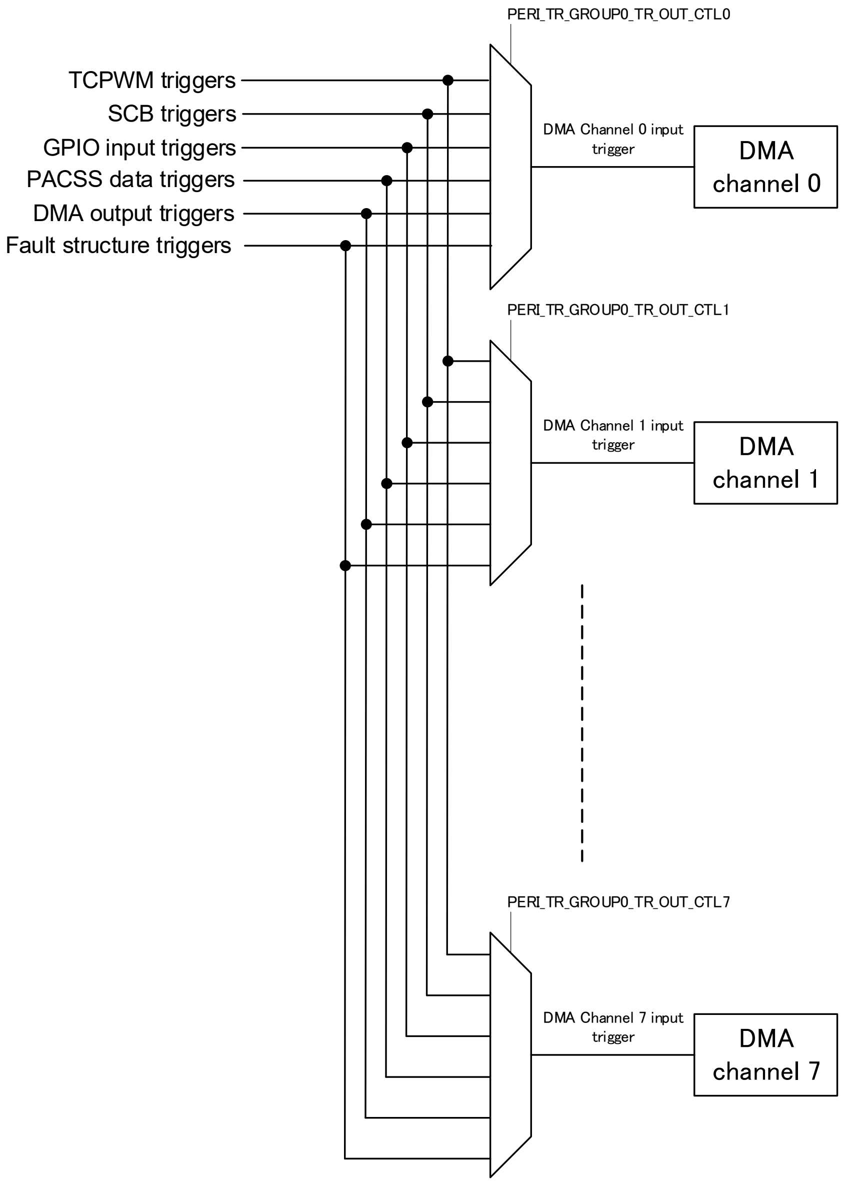

PERI_TR_GROUP0_TR_OUT_CTL0
TCPWM triggers
SCB triggers
GPIO input triggers
PACSS data triggers
DMA output triggers
Fault structure triggers
DMA Channel 0 input trigger
DMA channel 0
PERI_TR_GROUP0_TR_OUT_CTL1
DMA Channel 1 input trigger
DMA channel 1
PERI_TR_GROUP0_TR_OUT_CTL7
DMA Channel 7 input trigger
DMA channel 7

Figure 6-2. Trigger Multiplexer implementation

The trigger source for individual DMA channels is selected in the PERI_TR_GROUP0_TR_OUT_CTLx[5:0] register.

Table 6-1 and **Table 6-2** provide the PSoC™ 4 HV PA DMA trigger multiplexers and the multiplexer outputs.

Table 6-1. DMA Trigger sources

PERI_TR_GROUP0_TR_OUT_CTL x[5:0]	Trigger source
0	Software trigger
1	DMA Channel 0 trigger out
2	DMA Channel 1 trigger out
3	DMA Channel 2 trigger out
4	DMA Channel 3 trigger out
5	DMA Channel 4 trigger out
6	DMA Channel 5 trigger out
7	DMA Channel 6 trigger out
8	DMA Channel 7 trigger out
9	Fault structure output #0
10	Fault structure output #1
11	TCPWM 0 overflow
12	TCPWM 1 overflow
13	TCPWM 2 overflow
14	TCPWM 3 overflow
15	TCPWM 0 underflow
16	TCPWM 1 underflow
17	TCPWM 2 underflow
18	TCPWM 3 underflow
19	TCPWM 0 compare match
20	TCPWM 1 compare match
21	TCPWM 2 compare match
22	TCPWM 3 compare match
23	SCB 0 TX request
24	SCB 0 RX request
25	PACSS data valid channel 0
26	PACSS data valid channel 1
27	PACSS data valid channel 2
28	PACSS data valid channel 3
29	GPIO input trigger 0
30	GPIO input trigger 1
31	GPIO input trigger 2
32	GPIO input trigger 3

Table 6-2. Trigger Multiplexer outputs

Output	Trigger source
0	Triggers to DMA Channel 0
1	Triggers to DMA Channel 1
2	Triggers to DMA Channel 2
3	Triggers to DMA Channel 3
4	Triggers to DMA Channel 4
5	Triggers to DMA Channel 5
6	Triggers to DMA Channel 6
7	Triggers to DMA Channel 7

6.3.1 Creating software triggers

Every DMA channel has a trigger input and output trigger associated with it. This trigger input can come from any trigger group, as described in **"Trigger multiplexer"** on page 37. A software trigger for the DMA channel is implemented using the trigger input option 0 in the trigger multiplexer settings.
When PERI_TR_GROUP0_TR_OUT_CTLx [5:0] is zero, the DMA trigger is configured for a software trigger. The DMA channel is then triggered using the PERI_TR_CTL register.

6.3.2 Pending triggers

When a DMA channel is already operational and a trigger event is encountered, the DMA channel corresponding to the trigger is put into a pending state. Pending triggers keep track of activated triggers by locally storing them in pending bits. This is essential, because multiple channel triggers may be activated simultaneously, whereas only one channel can be served by the data transfer engine at a time. This block enables the use of both level-sensitive and pulse-sensitive triggers.

The pending triggers are registered in the status register (DMAC_STATUS_CH_ACT).

6.3.3 Output triggers

Each channel has an output trigger. This trigger is high for two system clock cycles. The trigger is generated on the completion of a data transfer. At the system level, these output triggers can be connected to the trigger multiplexer component. This connection allows for a DMA controller output trigger to be connected to a DMA controller input trigger. In other words, the completion of a transfer in one channel can activate another channel or even reactivate the same channel.

6.3.4 Channel prioritization

When there are multiple channels with active triggers, the channel priority is used to determine which channel gets the access to the data transfer engine. The priorities are set for each channel using the PRIO field of the channel control register (DMAC_CH_CTL), with '0' representing the highest priority and '3' representing the lowest priority. Priority decoding uses the channel priority to determine the highest priority activated channel. If multiple activated channels have the same highest priority, the channel with the lowest index 'i', is considered the highest priority activated channel.

6.4 Data transfer engine

The data transfer engine is responsible for the data transfer from a source location to a destination location. When idle, the data transfer engine is ready to accept the highest priority activated channel. The configuration of the data transfer is specified by the descriptor. The data transfer engine implements a state machine, which has the following states.

- State 0 - Default State: This is the idle state of the DMA controller, where it waits for a trigger condition to initiate transfer.
- State 1 - Load Descriptor: When a trigger condition is encountered and priority is resolved, the data transfer engine enters the load descriptor state. In this state, the active descriptor (SRC, DST, and CTL) is loaded into the DMA controller to initiate the transfer. The DMAC_STATUS, DMAC_STATUS_SRC_ADDR and DMAC_STATUS_DST_ADDR, and STATUS_CH_ACT will also reflect the currently active status.
- State 2 - Loading data from source: The data transfer engine uses the master I/F to load data from the source location.
- State 3 - Storing data at destination: The data transfer engine uses the master I/F to store data to the destination location.
- Depending on the Transfer mode, State 2 and 3 may be performed multiple times.
- State 4 - Storing Descriptor: The data transfer engine updates the channel's descriptor structure to reflect the data transfer and stores it in the descriptor.
- State 5 - Wait for Trigger Deactivation: If the trigger deactivation condition is specified as two cycles, this condition is met after two cycles of the trigger activation. If it was set to 'wait indefinitely', the DMA controller will remain in this state until the trigger signal has gone low.
- State 6 - Storing Descriptor Response: In this phase, the data transfer according to the descriptor is completed and an interrupt may be generated if it was configured to do so. The Response field in DMAC_DESCR_PING_STATUS or DMAC_DESCR_PONG_STATUS is also populated and the state transitions to State 0.

6.5 Descriptors

The data transfer between a source and a destination in a channel is configured using a descriptor. Each channel in the DMA has two descriptors named PING and PONG descriptors (also called Descriptor 0 and Descriptor 1 in this document). A descriptor is a set of four 32-bit registers that contain the configuration for the transfer in the associated channel.

Figure 6-3 shows the structure of a descriptor.

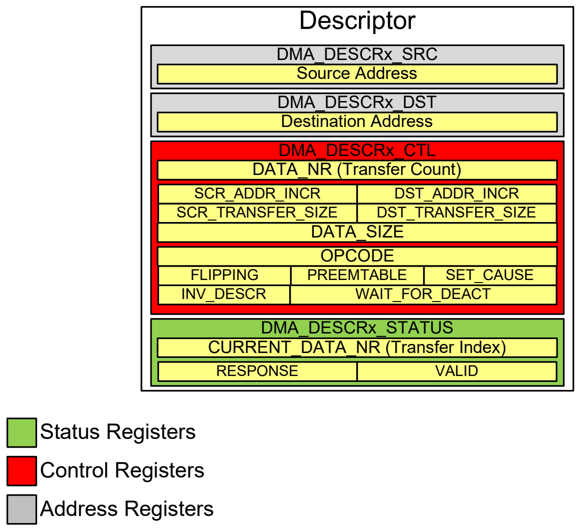


Figure 6-3. Descriptor Structure

6.6 Address configuration

Figure 6-4 demonstrates the use of the descriptor settings for the address configuration of a transfer.

Source and Destination Address: The Source and Destination addresses are set in the respective registers in the descriptor. These set the base addresses for the source and destination location for the transfer. In case the descriptor is configured to transfer a single element, this field holds the source/destination address of the data element. If the descriptor is configured to transfer multiple elements with source address or destination address or both in an incremental mode, this field will hold the address of the first element that is transferred.

Data Number (DATA_NR): This is a transfer count parameter. DATA_NR is a 16-bit number, which determines the number of elements to be transferred before a descriptor is defined as completed. In a typical use case, this setting is the buffer size of a transfer.

Source Address Increment (SCR_ADDR_INC): This is a bit setting in the control register, which determines if a source address is incremented between each data element transfer. This feature is enabled when the source of the data is a buffer and each transfer element needs to be fetched from subsequent locations in the memory. In this case, the Source Address register sets only the base address and subsequent transfers are incremental on this. The size of address increments are determined based on the SCR_TRANSFER_SIZE setting described in **"Transfer size"** on page 43.

Destination Address Increment (DST_ADDR_INC): This is a bit setting in the control register, which determines if a destination address is incremented between each element transfer. This feature is enabled when the destination of the data is a buffer and each transfer element needs to be transferred to subsequent locations in the memory. In this case, the Destination Address register sets only the base address and subsequent transfers are incremental on this. The size of address increments are determined based on the DST_TRANSFER_SIZE setting described in **"Transfer size"** on page 43.

Invalidate Descriptor (INV_DESCR): When this bit is set, the descriptor transfers all data elements and clears the descriptor's VALID bit, making it invalid. This feature affects the VALID bit in the DMA_DESCRx_STATUS register. This setting is used in cases where the user expects the descriptor to get invalidated after its transfer is complete. The descriptor can be made valid again in firmware by setting the VALID bit in the descriptor's STATUS register.

Preemptable (PREEMPTABLE): If disabled, the current transfer as defined by Operational mode is allowed to complete undisturbed. If enabled, the current transfer as defined by Operation Mode can be preempted/interrupted by a DMA channel of higher priority. When this channel is preempted, it is set as pending and will run the next time its priority is the highest.

Setting Interrupt Cause (SET_CAUSE): When the descriptor completes transferring all data elements, it generates an interrupt request. This interrupt request is shared among all DMA channels. Setting this bit enables the corresponding channel to be a source of this interrupt.

Trigger Type (WAIT_FOR_DEACT): When the DMA transfer based on the descriptor is completed, the data transfer engine checks the state of trigger deactivation. This is corresponding to State 5 of the data transfer engine. See **"Data transfer engine"** on page 40. The type of DMA input trigger will determine when the trigger signal is considered deactivated. The DMA transfer is activated when the trigger is activated, but the transfer is not considered complete until the trigger state is deactivated. This field is used to synchronize the controller's data transfer(s) with the agent that generated the trigger.

This field is ONLY used on completion of a descriptor execution and has four settings:

- 0 - Pulse Trigger: Do not wait for deactivation.
- 1 - Level-sensitive waits four SYSCLK cycles: The DMA trigger is deactivated after the level trigger signal is detected for four cycles.
- 2 - Level-sensitive waits eight SYSCLK cycles: The DMA transfer is initiated after the level trigger signal is detected for eight cycles.
- 3 - Pulse trigger waits indefinitely for deactivation. The DMA transfer is initiated after the trigger signal deactivates.

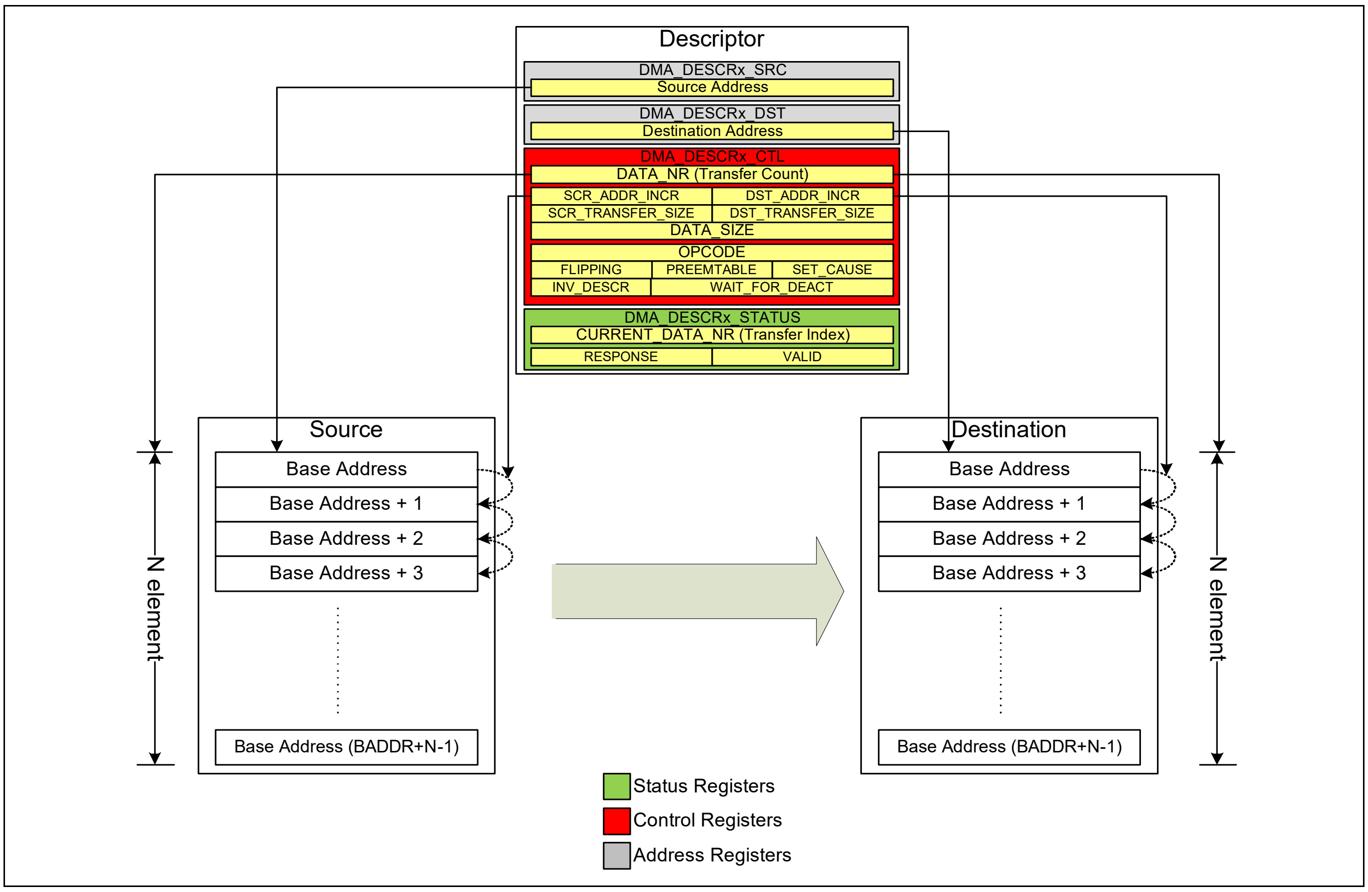


Figure 6-4. DMA Transfer: Address Configuration

6.7 Transfer size

The transfer word width for a transfer can be configured using the transfer/data size parameter in the descriptor. The settings are diversified into source transfer size, destination transfer size, and data size. The data size parameter (DATA_SIZE) sets the width of the bus for the transfer. The source and destination transfer sizes, set by SCR_TRANSFER_SIZE and DST_TRANSFER_SIZE, can have a value of either the DATA_SIZE or 32 bit. DATA_SIZE can be set to a 32-bit, 16-bit, or 8-bit setting.

The data width of most PSoC™ 4 HV PA peripheral registers is 4 bytes (32 bit); therefore, SCR_TRANSFER_SIZE or DST_TRANSFER_SIZE should typically be set to 32 bit when DMA is using a peripheral as its source or destination. The source and destination transfer size for the DMA component must match the addressable width of the source and destination, regardless of the width of data that needs to be moved. The DATA_SIZE parameter will correspond to the width of the actual data. For example, if a 16-bit PWM is used as a destination for DMA data, the DST_TRANSFER_SIZE must be set to 32 bit to match the width of the PWM register, because the peripheral register width for the TCPWM block (and most PSoC™ 4 HV PA peripherals) is always 32 bit. However, in this example the DATA_SIZE for the destination may still be set to 16 bit because the 16-bit PWM only uses 2 bytes of data. SRAM and flash are 8-bit, 16-bit, or 32-bit addressable and can use any source and destination transfer sizes to match the needs of the application.

Table 6-3 summarizes the possible combinations of the transfer size settings and its description.

Table 6-3. Transfer Size settings

DATA_SIZE	SCR_TRANSFER_SIZE	DST_TRANSFER_SIZE	Typical usage	Description
8-bit	8-bit	8-bit	Memory to Memory	No data manipulation
8-bit	32-bit	8-bit	Peripheral to Memory	Higher 24 bits from the source dropped
8-bit	8-bit	32-bit	Memory to Peripheral	Higher 24 bits zero padded at destination
8-bit	32-bit	32-bit	Peripheral to Peripheral	Higher 24 bits from the source dropped and higher 24 bits zero padded at destination
16-bit	16-bit	16-bit	Memory to Memory	No data manipulation
16-bit	32-bit	16-bit	Peripheral to Memory	Higher 16 bits from the source dropped
16-bit	16-bit	32-bit	Memory to Peripheral	Higher 16 bits zero padded at destination
16-bit	32-bit	32-bit	Peripheral to Peripheral	Higher 16 bits from the source dropped and higher 16-bit zero padded at destination
32-bit	32-bit	32-bit	Peripheral to Peripheral	No data manipulation
32-bit	32-bit	32-bit	Peripheral to Memory	No data manipulation

6.8 Descriptor Chaining

Every channel has a PING and PONG descriptor, which can have a distinct setting for the associated transfer. The active descriptor is set by the PING_PONG bit in the individual channel control register (DMAC_CH_CTL). The functionality of the PING and PONG descriptors is to create a link list of descriptors. This helps create a transition from one transfer configuration to another without CPU intervention. In addition, the two descriptors mean that the CPU is free to modify the PING register when PONG register is active and vice versa.

The FLIPPING bit in a descriptor, when enabled, links it to its PING/PONG counterpart. This field is used in conjunction with the OPCODE 2 transfer mode. Therefore, when the FLIPPING bit is enabled in a PING descriptor, configured for OPCODE 2, the channel automatically executes the PONG descriptor at the end of the PING descriptor. In case the configuration is for an OPCODE 0 or OPCODE 1, a new trigger is required to start the PONG descriptor.

The use of PING PONG has more relevance in the context of transfer modes.

6.9 Transfer modes

The operation of a channel during the execution of a descriptor is defined by the OPCODE settings. Three OPCODEs are possible for each channel of the DMA controller.

6.9.1 Single data element per trigger (OPCODE 0)

This mode is achieved when an OPCODE of 0 is configured. DMA transfers a single data element from a source location to a destination location on each trigger signal. This functionality can be used in conjunction with other settings in the descriptor such as Source and Destination increment.

Figure 6-5 shows a typical use case of this transfer. Here a UART receive (RX) register is the source and the destination is a peripheral register such as an SPI transmit (TX) register. The trigger is from the DMA request signal of the UART. When the trigger is received, the transfer engine will load data from the UART RX register and store the lower eight bits to the SPI TX register. Successive triggers will result in the same behavior because the descriptor will be rerun.

Note how the source and destination data widths are assigned as 32 bit. This is because all accesses to peripheral registers in PSoC™ must be 32 bit. Because the valid data width is only eight bits, the DATA_SIZE is maintained as eight bit.

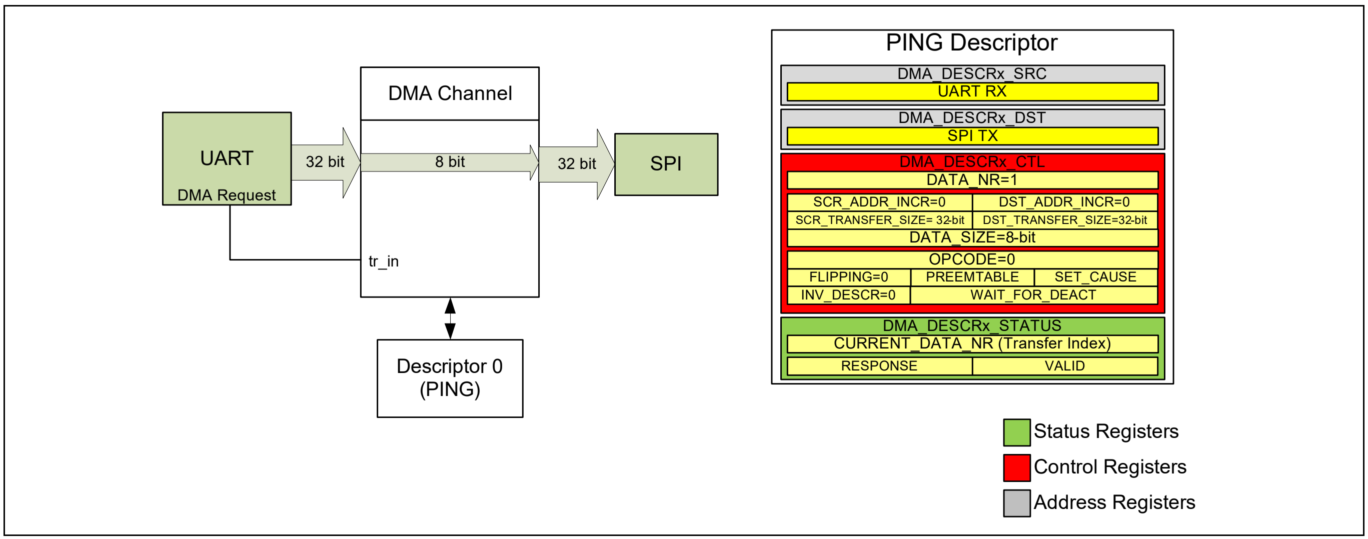


Figure 6-5. OPCODE 0: Simple DMA Transfer from Peripheral to Peripheral

Figure 6-6 describes another use case where the data transfer is between the UART RX register and a buffer. The use case shows a PING descriptor, which is configured to increment the destination while taking data from a source location, which is a UART. When the trigger is received, the transfer engine will load data from the UART RX register and store to the Memory Buffer, Sample 1 memory location. Subsequent triggers will continue to store the UART data into consecutive locations from Sample 1, until the PING descriptor buffer size (DATA_NR field) is filled.

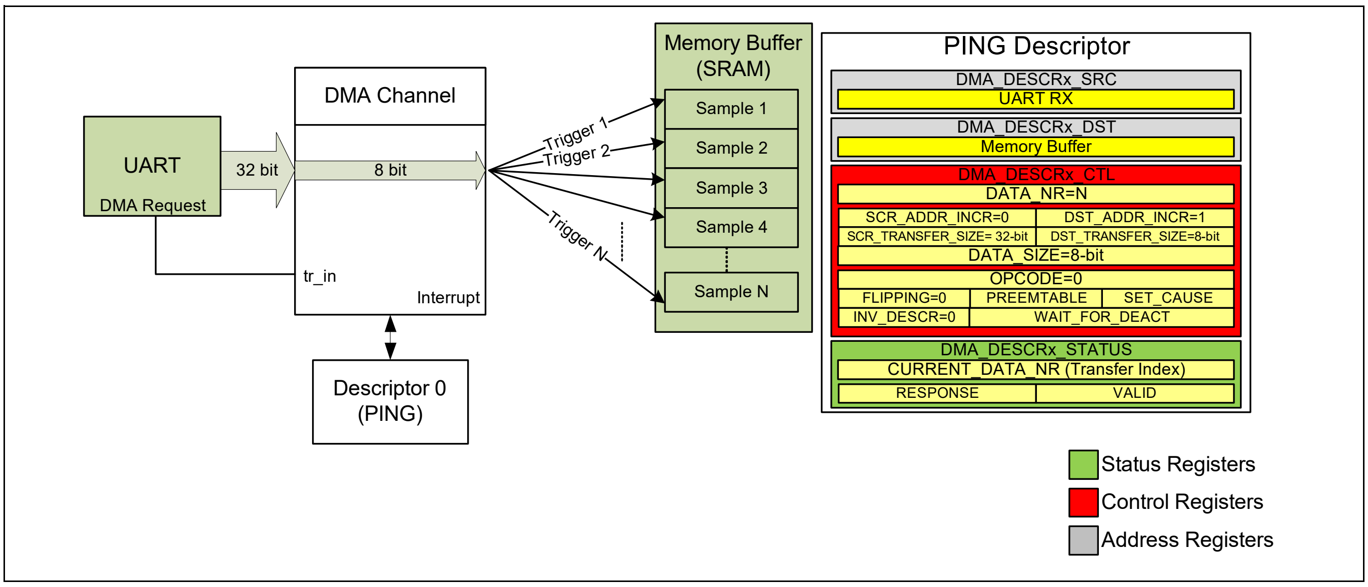


Figure 6-6. OPCODE 0: Transfer with Destination Address Increment feature

A similar use case is shown in **Figure 6-7**. This demonstrates the use of the PING and PONG descriptors. On completion of the PING descriptor, the controller will flip to execute the PONG descriptor. Thus, two buffer transfers are achieved in sequence. However, note that the transfers are still done at one element transfer for every trigger.

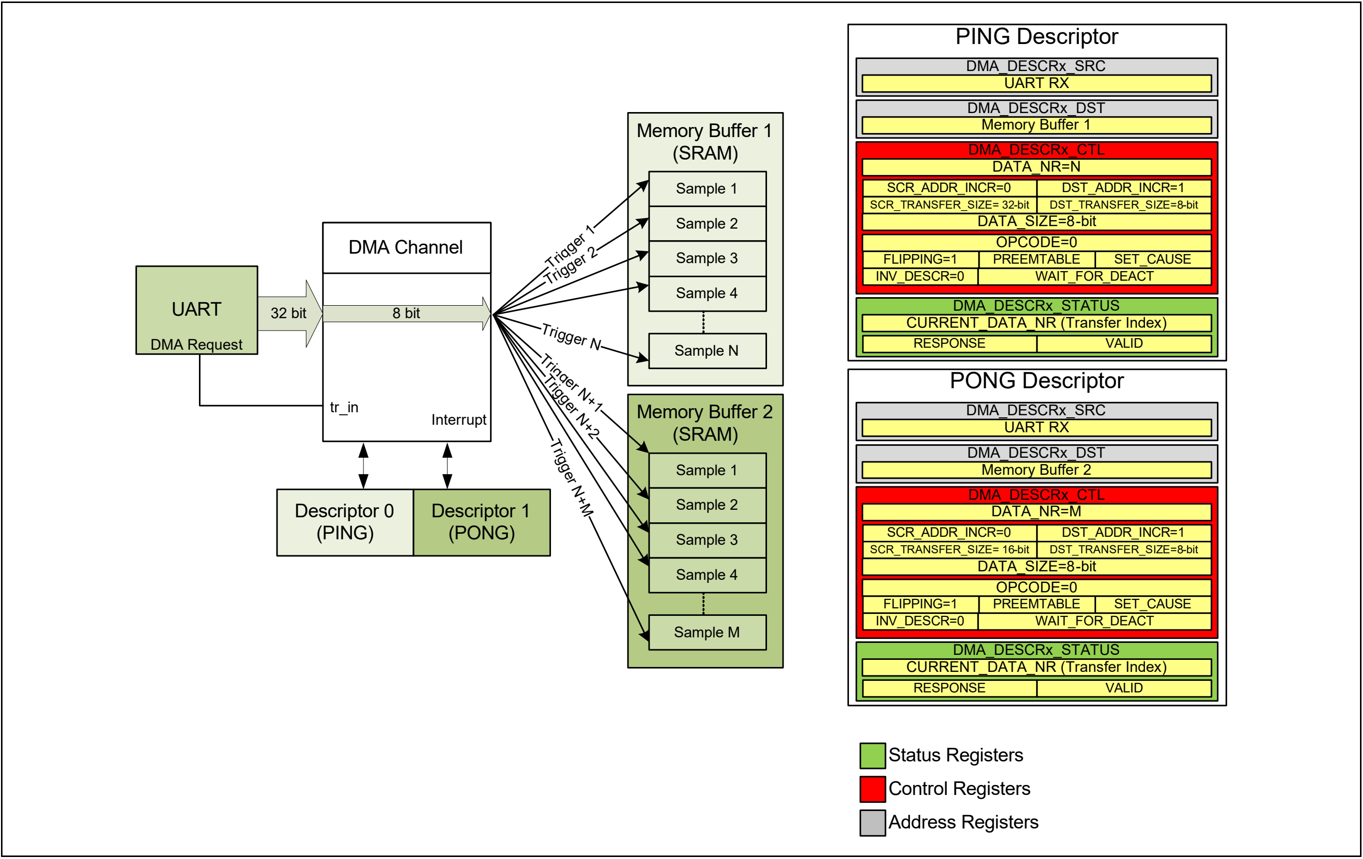


Figure 6-7. DMA Transfer using Flipping feature

6.9.2 Entire descriptor per trigger (OPCODE 1)

In this mode of operation, the DMA transfers multiple data elements from a source location to a destination location in one trigger. In OPCODE 1, the controller executes the entire descriptor in a single trigger. This type of functionality is useful in memory-to-memory buffer transfers. When the trigger condition is encountered, the transfer is continued until the descriptor is completed.

Figure 6-8 shows an OPCODE 1 transfer, which transfers the entire contents of the source buffer into the destination buffer. The entire transfer is part of a single PING descriptor and is completed on a single trigger.

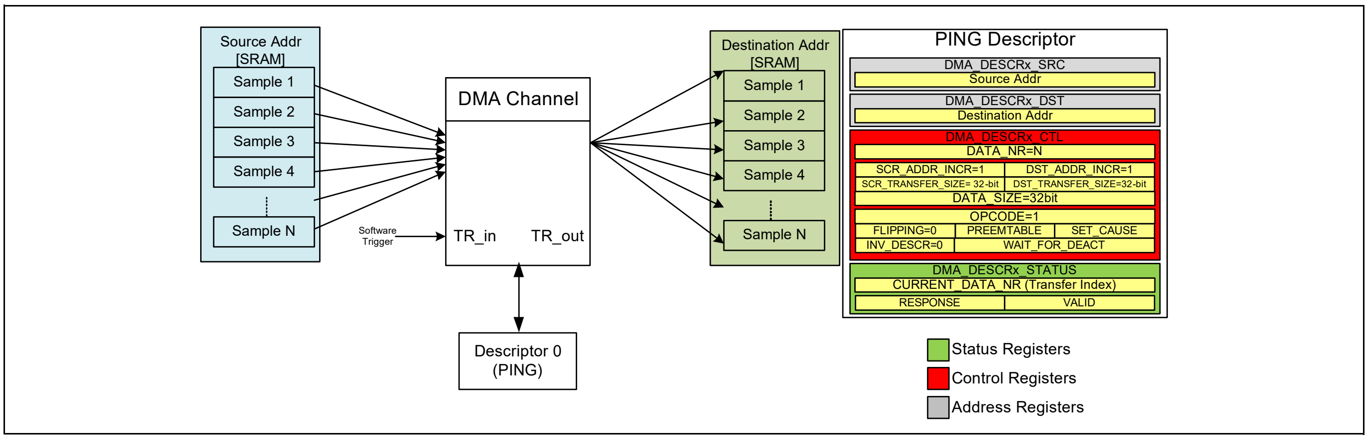


Figure 6-8. DMA Transfer Example with OPCODE 1

6.9.3 Entire descriptor chain per trigger (OPCODE 2)

OPCODE 2 is always used in conjunction with the FLIPPING field. When OPCODE 2 is used with FLIPPING enabled in a PING descriptor, a single trigger can execute a PING descriptor and automatically flip to the PONG descriptor and execute that too. If the PONG descriptor is also provided with an OPCODE 2, then the cycling between PING and PONG will continue until one of the descriptors are invalidated or changed by the CPU.

Figure 6-9 shows a case where the PING and PONG descriptors are configured for OPCODE 2 operation and on the second iteration of the PING register, FLIPPING is disabled by the CPU.

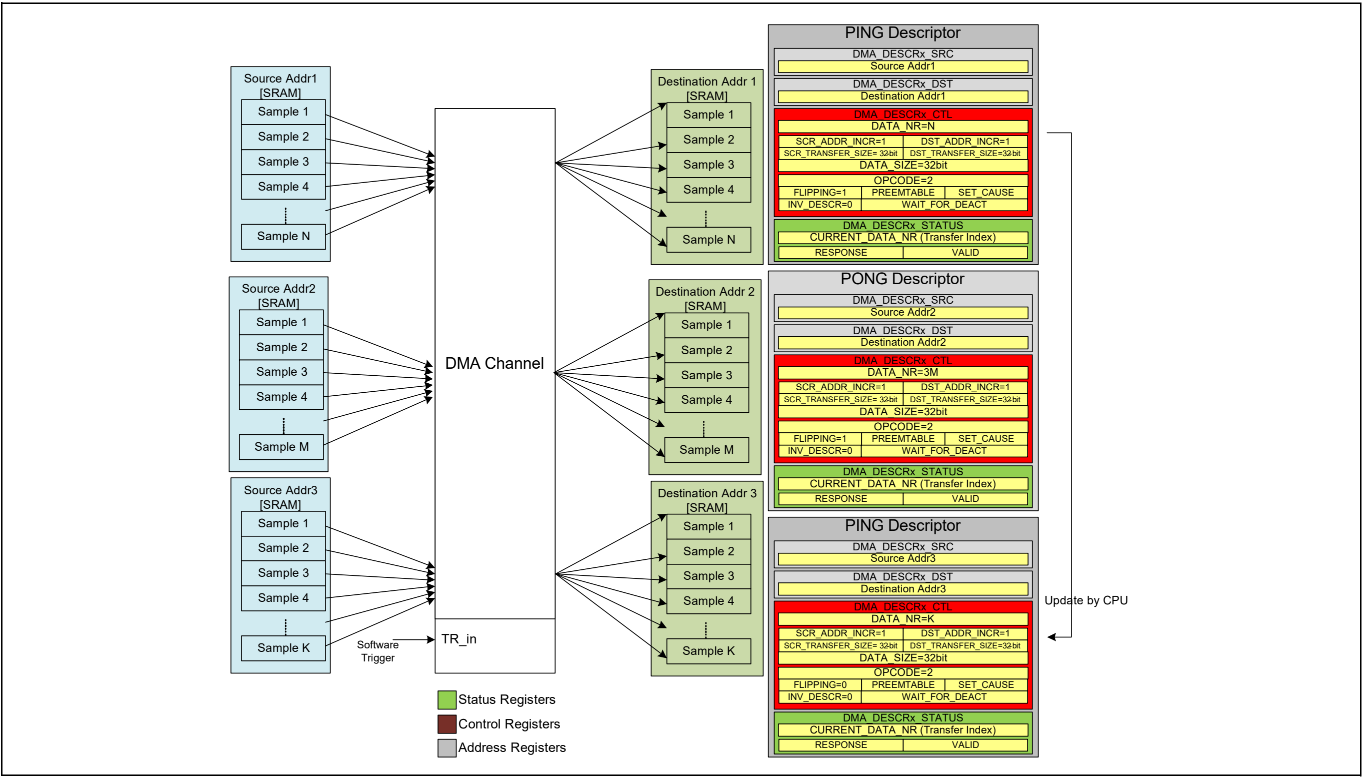


Figure 6-9. DMA Transfer Example with OPCODE 2

The OPCODE 2 transfer mode can be customized to implement distinct use cases. Figure 6-10 illustrates one such use case. Here, the source data can come from two different locations which are not consecutive memory. The destination is a data structure that is in consecutive memory locations. One source is Timer 2, which holds a timing data and the other source is a PWM compare register. Both the data is stored in consecutive locations in memory.

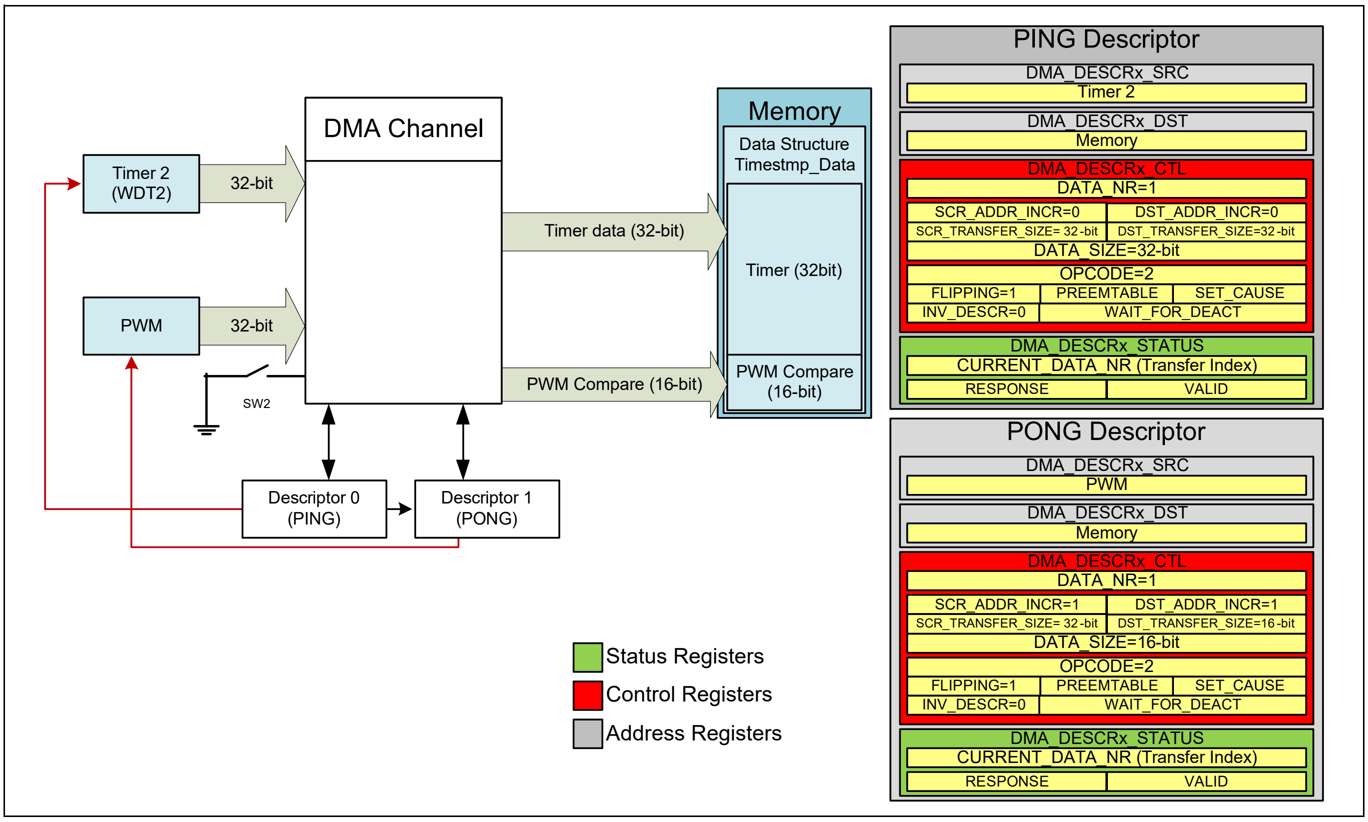


Figure 6-10. OPCODE 2: Multiple sources to Memory

6.10 Operation and timing

Figure 6-11 shows the DMA controller design with a trigger, data, or interrupt flow superimposed on it.

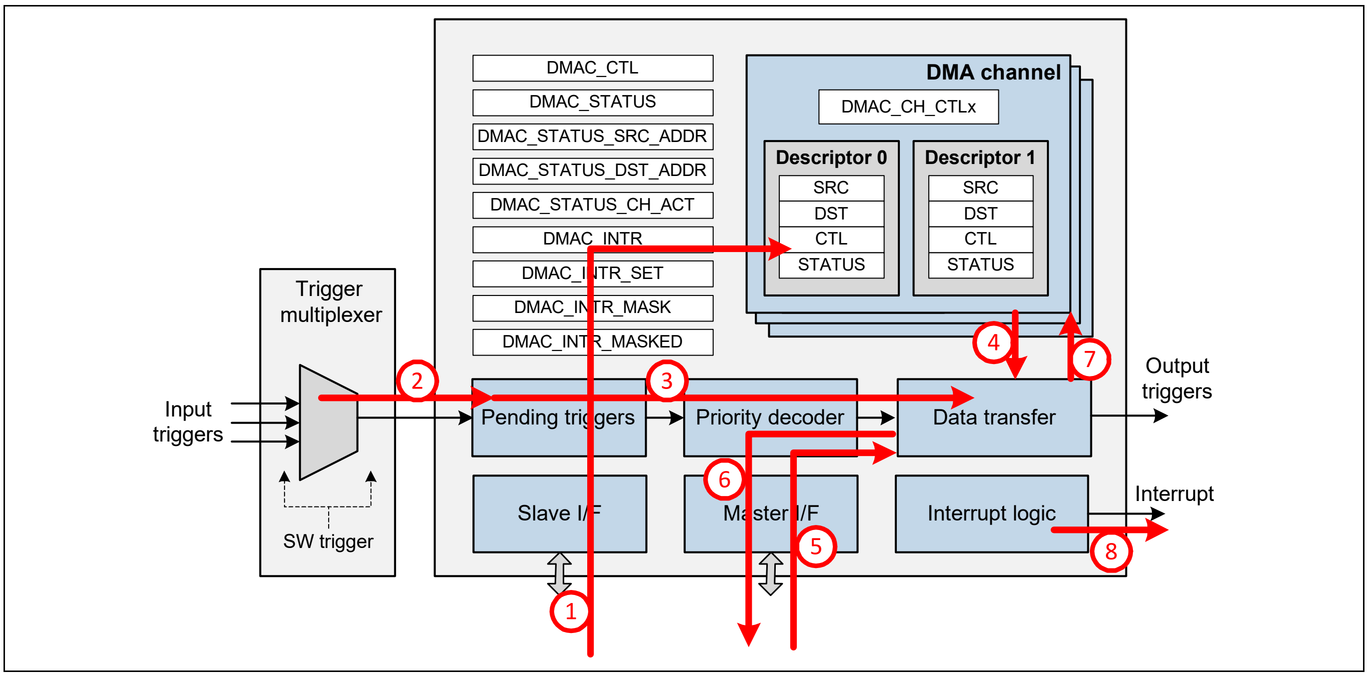


Figure 6-11. Operational flow

The flow exemplifies the steps that are involved in a DMA controller data transfer:

1. The main CPU programs the descriptor structure for a specific channel. It also programs the DMA register that selects a specific system trigger for the channel.
2. The channel's system trigger is activated.
3. Priority decoding determines the highest priority activated channel.
4. The data transfer engine accepts the activated channel and uses the channel identifier to load the channel's descriptor structure. The descriptor structure specifies the channel's data transfers.
5. The data transfer engine uses the master I/F to load data from the source location.
6. The data transfer engine uses the master I/F to store data to the destination location. In a single element (opcode 0) transfer, steps 5 and 6 are performed once. In a multiple element descriptor (opcode 1 or 2) transfer, steps 5 and 6 may be performed multiple times in sequence to implement multiple data element transfers.
7. The data transfer engine updates the channel's descriptor structure to reflect the data transfer and stores it in the descriptor SRAM.
8. If all the data transfers as specified by a descriptor channel structure have completed, an interrupt may be generated (this is a programmable option).

The DMA controller data transfer steps can be classified as either: initialization, concurrent, or sequential steps:

- Initialization: This includes step 1, which programs the descriptor structures. This step is done for each descriptor structure. It is performed by the main CPU and is NOT initiated by an activated channel trigger.
- Concurrent: This includes steps 2 and 3. These steps are performed in parallel for each channel.
- Sequential: This includes steps 4 through 8. These steps are performed sequentially for each activated channel. As a result, the DMA controller throughput is determined by the time it takes to perform these steps. This time consists of two parts: the time spent by the controller (to load and store the descriptor) and the time spent on the bus infrastructure. The latter time is dependent on the latency of the bus (determined by arbiter and bridge components) and the target memories/peripherals.

When transferring single data elements, it takes 12 clock cycles to complete one full transfer under the assumption of no wait states on the AHB-Lite bus. The equation for number of cycles to complete a transfer in this mode is:

No. of cycles = 12 + LOAD wait states + STORE wait states

When transferring entire descriptors or chaining descriptor chains, 12 clock cycles are needed for the first data element. Subsequent elements need three cycles. This is also under the assumption of no wait states on the AHB-Lite bus. The equation for number of cycles to transfer 'N' elements is:

No. of cycles = (12 + LOAD wait states + STORE wait states) + (N–1) × (3 + LOAD wait states + STORE wait states)

6.11 Arbitration

The AHB bus of the device has two masters: the CPU and the DMA controller. All peripherals and memory connect to the bus through slave interfaces. There are dedicated slave interfaces for flash memory and RAM with their own arbiters. The peripheral registers all connect to a single slave interface through a bridge into a dedicated arbiter. The DMA controller's slave interface, which is used to access the DMA controller's control registers, all connect through another slave interface. **Figure 6-12** illustrates this architecture.

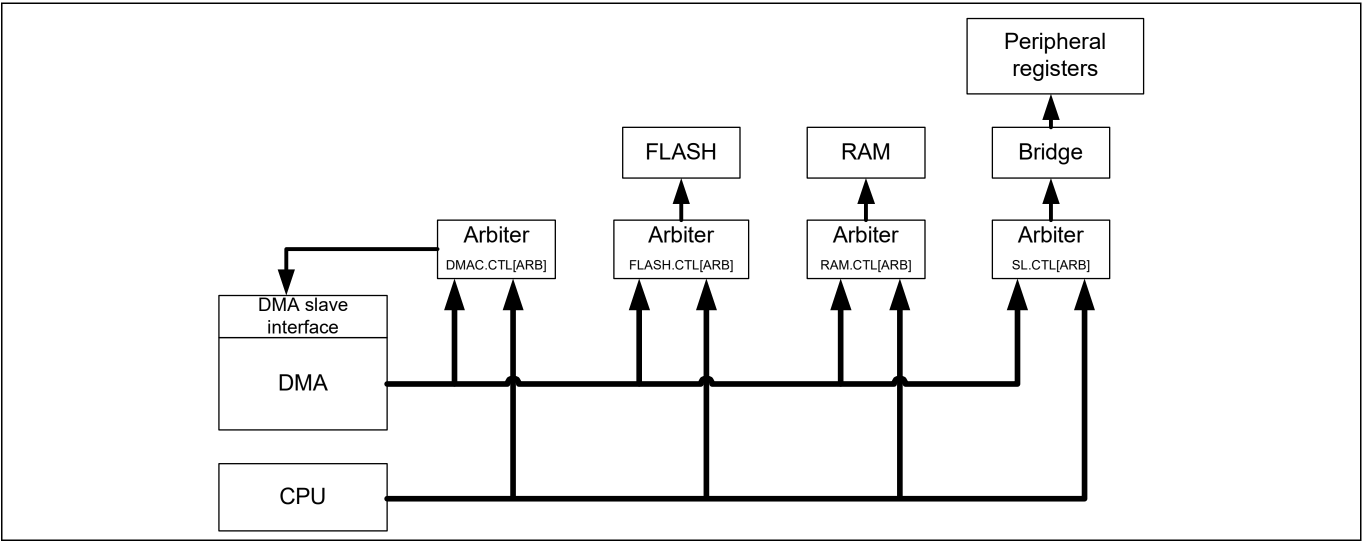


Figure 6-12. PSoC™ Bus Architecture

The arbitration policy for each slave can be one of the following:

- CPU priority: CPU always has the priority on arbitration. DMA access is allowed only when there are no CPU requests.
- DMA priority: DMA always has the priority on arbitration. CPU access is allowed only when there are no DMA requests.
- Round-robin: The arbitration priority keeps switching between DMA and CPU for every request. The arbitration priority switches for every request – CPU or DMA.
- Round-robin sticky: This mode is similar to the round robin, but the priority switches only when there is a request from lower priority master. For example, if the current priority was CPU and there was a request made by the DMA, the priority switches to DMA for the next request. If there was no request from DMA, CPU holds the current priority.

The arbitration models are illustrated using the following diagrams.

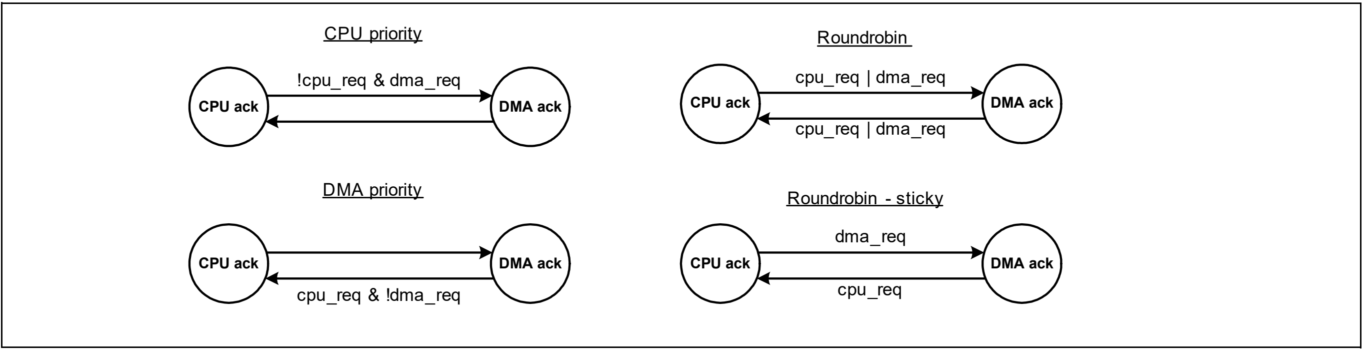


Figure 6-13. Arbitration models

6.12 Register list

Table 6-4. Register list

Register name	Comments	Features
DMACv3		
DMAC_CTL	Block Control Register	Enable bit for the DMA controller.
DMAC_STATUS	Block Status Register	Provides status information of the DMA controller.
DMAC_STATUS_SRC_ADDR	Current Source Address Register	Provides details of the source address currently being loaded.
DMAC_STATUS_DST_ADDR	Current Destination Address Register	Provides details of the destination address currently being loaded.
DMAC_STATUS_CH_ACT	Channel Activation Status Register	Software reads this field to get information on all actively pending channels (either in pending or in the data transfer engine).
DMAC_CH_CTLx	Channel Control Register	Provides channel enable, PING/PONG and priority settings for Channel x.
DMAC_DESCRx_PING_SRC	PING Source Address Register	Base address of source location for Channel x.
DMAC_DESCRx_PING_DST	PING Destination Address Register	Base address of destination location for Channel x.
DMAC_DESCRx_PING_CTL	PING Control Word Register	All control settings for the PING descriptor.
DMAC_DESCRx_PING_STATUS	PING Status Word Register	Validity, response, and real time Data_NR index status.
DMAC_DESCRx_PONG_SRC	PONG Source Address Register	Base address of source location for Channel x.
DMAC_DESCRx_PONG_DST	PONG Destination Address Register	Base address of destination location for Channel x.
DMAC_DESCRx_PONG_CTL	PONG Control Word Register	All control settings for the PONG descriptor.

Table 6-4. Register list (continued)

DMAC_DESCRx_PONG_STATUS	PONG Status Word Register	Validity, response, and real time Data_NR index status.
DMAC_INTR	Interrupt Register	The register fields are not retained. This is to ensure that they come up as '0' after coming out of Deep Sleep system power mode.
DMAC_INTR_SET	Interrupt Set Register	When read, this register reflects the interrupt request register.
DMAC_INTR_MASK	Interrupt Mask Register	Mask for corresponding field in INTR register.
DMAC_INTR_MASKED	Interrupt Masked Register	When read, this register reflects a bit-wise and between the interrupt request and mask registers. This register allows the software to read the status of all mask-enabled interrupt causes with a single load operation, rather than two load operations: one for the interrupt causes and one for the masks. This simplifies firmware development.
PERI		
PERI_TR_CTL	Trigger Control Register	This register provides SW control over trigger activation.
PERI_TR_GROUP0_TR_OUT_CTLx	Trigger Control Register	This register specifies the input trigger for a specific output trigger in trigger group0.

7 Interrupts

The Arm® Cortex®-M0+ (CM0+) CPU in PSoC™ 4 HV PA supports interrupts and exceptions. Interrupts refer to those events generated by peripherals external to the CPU such as timers, serial communication block, and port pin signals. Exceptions refer to those events that are generated by the CPU such as memory access faults and internal system timer events. Both interrupts and exceptions result in the current program flow being stopped and the exception handler or interrupt service routine (ISR) being executed by the CPU. The device provides a unified exception vector table for both interrupt handlers/ISR and exception handlers.

7.1 Features

PSoC™ 4 HV PA supports the following interrupt features:

- Supports 32 interrupts
- Nested vectored interrupt controller (NVIC) integrated with CPU core, yielding low interrupt latency
- Vector table may be placed in either flash or SRAM
- Configurable priority levels from 0 to 3 for each interrupt
- Level-triggered and pulse-triggered interrupt signals

7.2 How it works

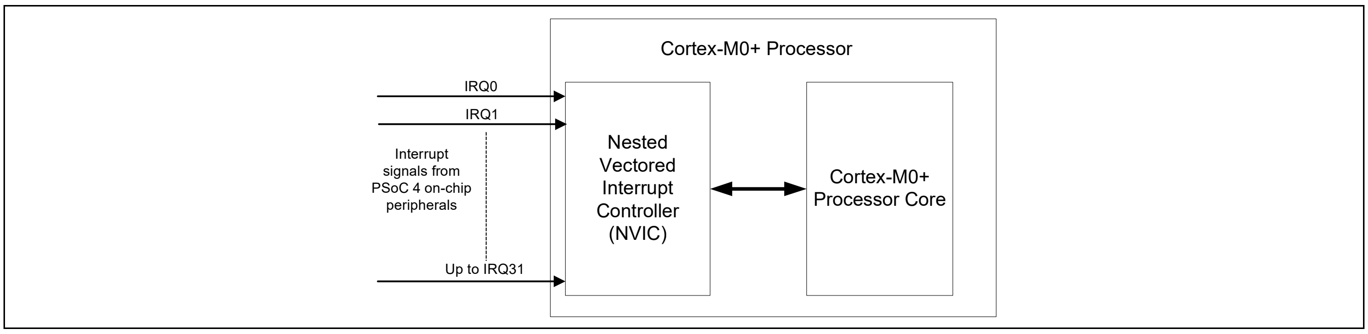

Cortex-M0+ Processor
IRQ0
IRQ1
Interrupt signals from PSoC 4 on-chip peripherals
Up to IRQ31
Nested Vectored Interrupt Controller (NVIC)
Cortex-M0+ Processor Core

Figure 7-1. PSoC™ 4 HV PA Interrupts block diagram

Figure 7-1 shows the interaction between interrupt signals and the Cortex®-M0+ CPU. PSoC™ 4 HV PA has up to 32 interrupts; these interrupt signals are processed by the NVIC. The NVIC takes care of enabling/disabling individual interrupts, priority resolution, and communication with the CPU core. The exceptions are not shown in **Figure 7-1** because they are part of CM0+ core generated events, unlike interrupts, which are generated by peripherals external to the CPU.

7.3 Interrupts and exceptions - Operation

7.3.1 Interrupt/exception handling

The following sequence of events occurs when an interrupt or exception event is triggered:

1. Assuming that all the interrupt signals are initially low (idle or inactive state) and the processor is executing the main code, a rising edge on any one of the interrupt lines is registered by the NVIC. The interrupt line is now in a pending state waiting to be serviced by the CPU.
2. On detecting the interrupt request signal from the NVIC, the CPU stores its current context by pushing the contents of the CPU registers onto the stack.
3. The CPU also receives the exception number of the triggered interrupt from the NVIC. All interrupts and exceptions have a unique exception number, as given in **Table 7-1**. By using this exception number, the CPU fetches the address of the specific exception handler from the vector table.
4. The CPU then branches to this address and executes the exception handler that follows.
5. Upon completion of the exception handler, the CPU registers are restored to their original state using stack pop operations; the CPU resumes the main code execution.

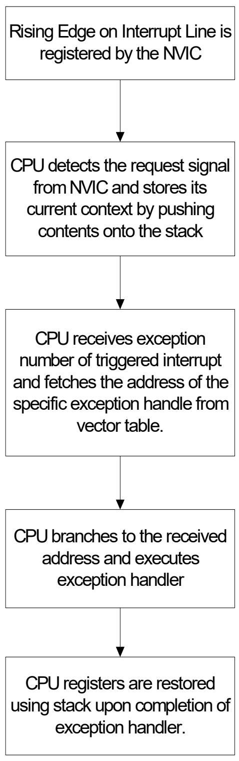

Rising Edge on Interrupt Line is registered by the NVIC
CPU detects the request signal from NVIC and stores its current context by pushing contents onto the stack
CPU receives exception number of triggered interrupt and fetches the address of the specific exception handle from vector table.
CPU branches to the received address and executes exception handler
CPU registers are restored using stack upon completion of exception handler.

Figure 7-2. Interrupt Handling when Triggered

When the NVIC receives an interrupt request while another interrupt is being serviced or receives multiple interrupt requests at the same time, it evaluates the priority of all these interrupts, sending the exception number of the highest priority interrupt to the CPU. Thus, a higher priority interrupt can block the execution of a lower priority ISR at any time.

Exceptions are handled in the same way that interrupts are handled. Each exception event has a unique exception number, which is used by the CPU to execute the appropriate exception handler.

7.3.2 Level and pulse interrupts

NVIC supports both level and pulse signals on the interrupt lines (IRQ0 to IRQ31). The classification of an interrupt as level or pulse is based on the interrupt source.

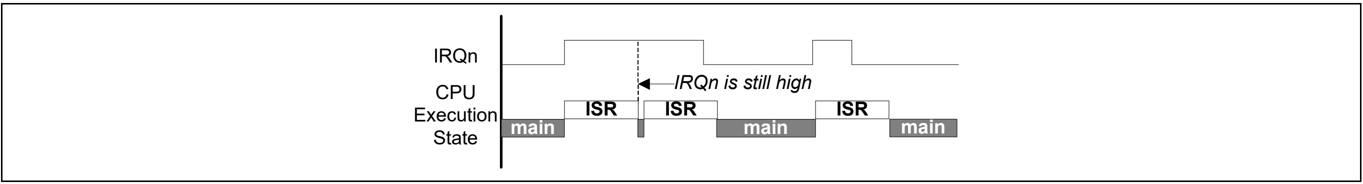


Figure 7-3. Level Interrupts

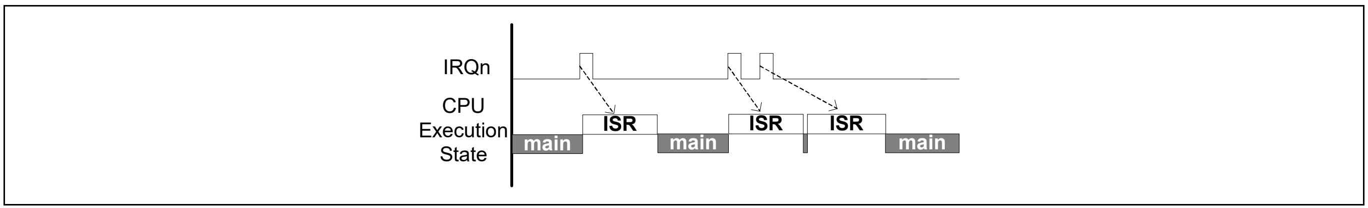


Figure 7-4. Pulse Interrupts

Figure 7-3 and **Figure 7-4** show the working of level and pulse interrupts, respectively. Assuming the interrupt signal is initially inactive (logic low), the following sequence of events explains the handling of level and pulse interrupts:

1. On a rising edge event of the interrupt signal, the NVIC registers the interrupt request. The interrupt is now in the pending state, which means the interrupt requests have not yet been serviced by the CPU.
2. The NVIC then sends the exception number along with the interrupt request signal to the CPU. When the CPU starts executing the ISR, the pending state of the interrupt is cleared.
3. When the ISR is being executed by the CPU, one or more rising edges of the interrupt signal are logged as a single pending request. The pending interrupt is serviced again after the current ISR execution is complete (see **Figure 7-4** for pulse interrupts).
4. If the interrupt signal is still high after completing the ISR, it will be pending and the ISR is executed again. **Figure 7-3** illustrates this for level triggered interrupts, where the ISR is executed as long as the interrupt signal is high.

7.3.3 Exception vector table

The exception vector table (**Table 7-1**), stores the entry point addresses for all exception handlers. The CPU fetches the appropriate address based on the exception number.

Table 7-1. Exception vector table

Exception number	Exception	Exception priority	Vector Address
–	Initial Stack Pointer Value	Not applicable (NA)	Base_Address - 0x00000000 (start of flash memory) or 0x20000000 (start of SRAM)
1	Reset	–3, the highest priority	Base_Address + 0x04
2	Non Maskable Interrupt (NMI)	–2	Base_Address + 0x08
3	HardFault	–1	Base_Address + 0x0C
4–10	Reserved	NA	Base_Address + 0x10 to Base_Address + 0x28
11	Supervisory Call (SVCall)	Configurable (0–3)	Base_Address + 0x2C
12–13	Reserved	NA	Base_Address + 0x30 to Base_Address + 0x34
14	PendSupervisory (PendSV)	Configurable (0–3)	Base_Address + 0x38
15	System Timer (SysTick)	Configurable (0–3)	Base_Address + 0x3C
16	External Interrupt(IRQ0)	Configurable (0–3)	Base_Address + 0x40
…	…	Configurable (0–3)	…
43	External Interrupt(IRQ31)	Configurable (0–3)	Base_Address + 0xBC

In **Table 7-1**, the first word (4 bytes) is not marked as exception number zero. This is because the first word in the exception table is used to initialize the main stack pointer (MSP) value on device reset; it is not considered as an exception. The vector table can be located anywhere in the memory map (flash or SRAM) by modifying the Vector Table Offset Register (VTOR). This register is part of the System Control Space of CM0+ located at 0xE000ED08. This register takes bits 31:8 of the vector table address; bits 7:0 are reserved. Therefore, the vector table address should be 256 bytes aligned. The advantage of moving the vector table to SRAM is that the exception handler addresses can be dynamically changed by modifying the SRAM vector table contents. However, the nonvolatile flash memory vector table must be modified by a flash memory write.

Reads of flash addresses 0x00000000 and 0x00000004 are redirected to the first eight bytes of SROM to fetch the stack pointer and reset vectors, unless the DIS_RESET_VECT_REL bit of the CPUSS_SYSREQ register is set. The default value of this bit at reset is 0 ensuring that reset vector is always fetched from SROM. To allow flash read from addresses 0x00000000 and 0x00000004, the DIS_RESET_VECT_REL bit should be set to ‘1’. The stack pointer vector holds the address that the stack pointer is loaded with on reset. The reset vector holds the address of the boot sequence. This mapping is done to use the default addresses for the stack pointer and reset vector from SROM when the device reset is released. For reset, boot code in SROM is executed first and then the CPU jumps to address 0x00000004 in flash to execute the handler in flash. The reset exception address in the SRAM vector table is never used.

Also, when the SYSCALL_REQ bit of the CPUSS_SYSREQ register is set, reads of flash address 0x00000008 are redirected to SROM to fetch the NMI vector address instead of from flash. Reset CPUSS_SYSREQ to read the flash at address 0x00000008.

The exception sources (exception numbers 1 to 15) are explained in **“Exception sources”** on page 58. The exceptions marked as Reserved in **Table 7-1** are not used, although they have addresses reserved for them in the vector table. The interrupt sources (exception numbers 16 to 35) are explained in **“Interrupt sources”** on page 59.

7.4 Exception sources

This section explains the different exception sources listed in **Table 7-1** (exception numbers 1 to 15).

7.4.1 Reset exception

Device reset is treated as an exception in PSoC™ 4 HV PA. It is always enabled with a fixed priority of –3, the highest priority exception. A device reset can occur due to multiple reasons, such as power-on-reset (POR), external reset signal on XRES pin, or watchdog reset. When the device is reset, the initial boot code for configuring the device is executed out of supervisory read-only memory (SROM). The boot code and other data in SROM memory are programmed by Infineon, and are not read/write accessible to external users. After completing the SROM boot sequence, the CPU code execution jumps to flash memory. Flash memory address 0x00000004 (Exception#1 in **Table 7-1**) stores the location of the startup code in flash memory. The CPU starts executing code out of this address. Note that the reset exception address in the SRAM vector table will never be used because the device comes out of reset with the flash vector table selected. The register configuration to select the SRAM vector table can be done only as part of the startup code in flash after the reset is de-asserted.

7.4.2 Non-maskable interrupt (NMI) exception

Non-maskable interrupt (NMI) is the highest priority exception other than reset. It is always enabled with a fixed priority of –2. There are two ways to trigger an NMI exception in the device:

- **NMI exception by setting NMIPENDSET bit (user NMI exception):** An NMI exception can be triggered in software by setting the NMIPENDSET bit in the interrupt control state register (CM0P_ICSR register). Setting this bit will execute the NMI handler pointed to by the active vector table (flash or SRAM vector table).
- **System Call NMI exception:** This exception is used for nonvolatile programming operations such as flash write operation and flash checksum operation. It is triggered by setting the SYSCALL_REQ bit in the CPUSS_SYSREQ register. An NMI exception triggered by SYSCALL_REQ bit always executes the NMI exception handler code that resides in SROM. Flash or SRAM exception vector table is not used for system call NMI exception. The NMI handler code in SROM is not read/write accessible because it contains nonvolatile programming routines that should not be modified by the user.

7.4.3 HardFault exception

HardFault is an always-enabled exception that occurs because of an error during normal or exception processing. HardFault has a fixed priority of –1, meaning it has higher priority than any exception with configurable priority. HardFault exception is a catch-all exception for different types of fault conditions, which include executing an undefined instruction and accessing an invalid memory addresses. The CM0+ CPU does not provide fault status information to the HardFault exception handler, but it does permit the handler to perform an exception return and continue execution in cases where software has the ability to recover from the fault situation.

7.4.4 Supervisor call (SVCall) exception

Supervisor Call (SVCall) is an always-enabled exception caused when the CPU executes the SVC instruction as part of the application code. Application software uses the SVC instruction to make a call to an underlying operating system and provide a service. This is known as a supervisor call. The SVC instruction enables the application to issue a supervisor call that requires privileged access to the system. Note that the CM0+ in PSoC™ 4 HV PA uses a privileged mode for the system call NMI exception, which is not related to the SVCall exception. (See the **"Chip operational modes"** on page 111 for details on privileged mode.) There is no other privileged mode support for SVCall at the architecture level in the device. The application developer must define the SVCall exception handler according to the end application requirements.

The priority of a SVCall exception can be configured to a value between 0 and 3 by writing to the two bit fields PRI_11[31:30] of the System Handler Priority Register 2 (SHPR2). When the SVC instruction is executed, the SVCall exception enters the pending state and waits to be serviced by the CPU. The SVCALLPENDED bit in the System Handler Control and State Register (SHCSR) can be used to check or modify the pending status of the SVCall exception.

7.4.5 PendSV exception

PendSV is another supervisor call related exception similar to SVCall, normally being software-generated. PendSV is always enabled and its priority is configurable. The PendSV exception is triggered by setting the PENDSVSET bit in the Interrupt Control State Register, CM0P_ICSR. On setting this bit, the PendSV exception enters the pending state, and waits to be serviced by the CPU. The pending state of a PendSV exception can be cleared by setting the PENDSVCLR bit in the Interrupt Control State Register, CM0P_ICSR. The priority of a PendSV exception can be configured to a value between 0 and 3 by writing to the two bit fields PRI_14[23:22] of the System Handler Priority Register 3 (CM0P_SHPR3). See the Armv6-M Architecture Reference Manual for more details.

7.4.6 SysTick exception

CM0+ CPU in PSoC™ 4 HV PA supports a system timer, referred to as SysTick, as part of its internal architecture. SysTick provides a simple, 24-bit decrementing counter for various timekeeping purposes such as an RTOS tick timer, high-speed alarm timer, or simple counter. The SysTick timer can be configured to generate an interrupt when its count value reaches zero, which is referred to as SysTick exception. The exception is enabled by setting the TICKINT bit in the SysTick Control and Status Register (CM0P_SYST_CSR). The priority of a SysTick exception can be configured to a value between 0 and 3 by writing to the two bit fields PRI_15[31:30] of the System Handler Priority Register 3 (SHPR3). The SysTick exception can always be generated in software at any instant by writing a one to the PENDSTSETb bit in the Interrupt Control State Register, CM0P_ICSR. Similarly, the pending state of the SysTick exception can be cleared by writing a one to the PENDSTCLR bit in the Interrupt Control State Register, CM0P_ICSR.

7.5 Interrupt sources

PSoC™ 4 HV PA supports up to 32 interrupts (IRQ0 to IRQ31 or exception numbers 16–47) from peripherals. The source of each interrupt is listed in **Table 7-2**. PSoC™ 4 HV PA provides flexible sourcing options for each interrupt line. The interrupts include standard interrupts from the on-chip peripherals such as TCPWM and serial communication block. The interrupt generated is usually the logical OR of the different peripheral states. The peripheral status register should be read in the ISR to detect which condition generated the interrupt. The interrupts are usually level interrupts, which require that the peripheral status register be read in the ISR to clear the interrupt. If the status register is not read in the ISR, the interrupt will remain asserted and the ISR will be executed continuously.

See the **"I/O system"** on page 141 for details on GPIO interrupts.

Table 7-2. List of PSoC™ 4 HV PA Interrupt Sources

Interrupt	Cortex®-M0+ Exception No.	Power Mode	Interrupt Source
NMI	2	Active	SYSCALL_REQ
IRQ0	16	Deep Sleep	SRSS interrupts
IRQ1	17	Deep Sleep	WDT interrupt
IRQ2	18	Deep Sleep	GPIO interrupt - All Port
IRQ3	19	Deep Sleep	GPIO interrupt - Port 0
IRQ4	20	Deep Sleep	GPIO interrupt - Port 1
IRQ5	21	–	Reserved
IRQ6	22	Deep Sleep	HVSS LIN interface interrupt
IRQ7	23	Deep Sleep	SCB0 (serial communication block 0)
IRQ8	24	Active	DMA interrupt
IRQ9	25	Active	SPCIF interrupt, controller 0
IRQ10	26	Active	SPCIF interrupt, controller 1
IRQ11	27	Active	Fault structure 0 interrupt
IRQ12	28	Active	Fault structure 1 interrupt
IRQ13	29	Active	LIN interrupt - channel 0
IRQ14	30	Active	LIN interrupt - channel 1
IRQ15	31	Active	TCPWM0 (Timer/Counter/PWM 0)
IRQ16	32	Active	TCPWM0 (Timer/Counter/PWM 1)
IRQ17	33	Active	TCPWM0 (Timer/Counter/PWM 2)
IRQ18	34	Active	TCPWM0 (Timer/Counter/PWM 3)
IRQ19	35	Active	PACSS consolidated interrupt
IRQ20	36	Active	PACSS digital channel 0
IRQ21	37	Active	PACSS digital channel 1
IRQ22	38	Active	PACSS digital channel 2
IRQ23	39	Active	PACSS digital channel 3
IRQ24	40	Active	PACSS MMIO
IRQ25–31	41–47	–	Reserved

7.6 Exception priority

Exception priority is useful for exception arbitration when there are multiple exceptions that need to be serviced by the CPU. PSoC™ 4 HV PA provides flexibility in choosing priority values for different exceptions. All exceptions other than Reset, NMI, and HardFault can be assigned a configurable priority level. The Reset, NMI, and HardFault exceptions have a fixed priority of –3, –2, and –1 respectively. In PSoC™ 4 HV PA, lower priority numbers represent higher priorities. This means that Reset, NMI, and HardFault exceptions have the highest priorities. The other exceptions can be assigned a configurable priority level between 0 and 3.

PSoC™ 4 HV PA supports nested exceptions in which a higher priority exception can obstruct (interrupt) the currently active exception handler. This pre-emption does not happen if the incoming exception priority is the same as active exception. The CPU resumes execution of the lower priority exception handler after servicing the higher priority exception. The CM0+ CPU in PSoC™ 4 HV PA allows nesting of up to four exceptions. When the CPU receives two or more exceptions requests of the same priority, the lowest exception number is serviced first.

The registers to configure the priority of exception numbers 1 to 15 are explained in **"Exception sources"** on page 58.

The priority of the 32 interrupts (IRQ0 to IRQ31) can be configured by writing to the Interrupt Priority registers (CM0P_IPR). This is a group of 32-bit registers with each register storing the priority values of four interrupts, as given in **Table 7-3**. The other bit fields in the register are not used.

Table 7-3. Interrupt Priority Register Bit Definitions

Bits	Name	Description
7:6	PRI_N0	Priority of interrupt number N.
15:14	PRI_N1	Priority of interrupt number N+1.
23:22	PRI_N2	Priority of interrupt number N+2.
31:30	PRI_N3	Priority of interrupt number N+3.

7.7 Enabling and disabling interrupts

The NVIC provides registers to individually enable and disable the 32 interrupts in software. If an interrupt is not enabled, the NVIC will not process the interrupt requests on that interrupt line. The Interrupt Set-Enable Register (CM0P_ISER) and the Interrupt Clear-Enable Register (CM0P_ICER) are used to enable and disable the interrupts respectively. These are 32-bit wide registers and each bit corresponds to the same numbered interrupt. These registers can also be read in software to get the enable status of the interrupts. **Table 7-4** shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 7-4. Interrupt Enable/Disable Registers

Register	Operation	Bit value	Comment
Interrupt Set Enable Register (CM0P_ISER)	Write	1	To enable the interrupt
		0	No effect
	Read	1	Interrupt is enabled
		0	Interrupt is disabled
Interrupt Clear Enable Register (CM0P_ICER)	Write	1	To disable the interrupt
		0	No effect
	Read	1	Interrupt is enabled
		0	Interrupt is disabled

The CM0P_ISER and CM0P_ICER registers are applicable only for interrupts IRQ0 to IRQ31. These registers cannot be used to enable or disable the exception numbers 1 to 15. The 15 exceptions have their own support for enabling and disabling, as explained in **"Exception sources"** on page 58.

The PRIMASK register in Cortex-M0+ (CM0+) CPU can be used as a global exception enable register to mask all the configurable priority exceptions irrespective of whether they are enabled. Configurable priority exceptions include all the exceptions except Reset, NMI, and HardFault listed in **Table 7-1**. They can be configured to a priority level between 0 and 3, 0 being the highest priority and 3 being the lowest priority. When the PM bit (bit 0) in the PRIMASK register is set, none of the configurable priority exceptions can be serviced by the CPU, though they can be in the pending state waiting to be serviced by the CPU after the PM bit is cleared.

7.8 Exception states

Each exception can be in one of the following states.

Table 7-5. Exception States

Exception State	Meaning
Inactive	The exception is not active or pending. Either the exception is disabled or the enabled exception is not triggered.
Pending	The exception request is received by the CPU/NVIC and the exception is waiting to be serviced by the CPU.
Active	An exception that is being serviced by the CPU but whose exception handler execution is not yet complete. A high-priority exception can interrupt the execution of lower priority exception. In this case, both the exceptions are in the active state.
Active and Pending	The exception is serviced by the processor and there is a pending request from the same source during its exception handler execution.

The Interrupt Control State Register (CM0P_ICSR) contains status bits describing the various exceptions states.

- The VECTACTIVE bits ([8:0]) in the CM0P_ICSR store the exception number for the current executing exception. This value is zero if the CPU does not execute any exception handler (CPU is in thread mode). Note that the value in VECTACTIVE bit fields is the same as the value in bits [8:0] of the Interrupt Program Status Register (IPSR), which is also used to store the active exception number.
- The VECTPENDING bits ([20:12]) in the CM0P_ICSR store the exception number of the highest priority pending exception. This value is zero if there are no pending exceptions.
- The ISRPENDING bit (bit 22) in the CM0P_ICSR indicates if a NVIC generated interrupt (IRQ0 to IRQ31) is in a pending state.

7.8.1 Pending exceptions

When a peripheral generates an interrupt request signal to the NVIC or an exception event occurs, the corresponding exception enters the pending state. When the CPU starts executing the corresponding exception handler routine, the exception is changed from the pending state to the active state.

The NVIC allows software pending of the 32 interrupt lines by providing separate register bits for setting and clearing the pending states of the interrupts. The Interrupt Set-Pending register (CM0P_ISPR) and the Interrupt Clear-Pending register (CM0P_ICPR) are used to set and clear the pending status of the interrupt lines. These are 32-bit wide registers and each bit corresponds to the same numbered interrupt.

Table 7-6 shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 7-6. Interrupt Set Pending/Clear Pending Registers

Register	Operation	Bit value	Comment
Interrupt Set-Pending Register (CM0P_ISPR)	Write	1	To put an interrupt to pending state
		0	No effect
	Read	1	Interrupt is pending
		0	Interrupt is not pending
Interrupt Clear-Pending Register (CM0P_ICPR)	Write	1	To clear a pending interrupt
		0	No effect
	Read	1	Interrupt is pending
		0	Interrupt is not pending

Setting the pending bit when the same bit is already set results in only one execution of the ISR. The pending bit can be updated regardless of whether the corresponding interrupt is enabled. If the interrupt is not enabled, the interrupt line will not move to the pending state until it is enabled by writing to the CM0P_ISER register.

Note that the CM0P_ISPR and CM0P_ICPR registers are used only for the 32 peripheral interrupts (exception numbers 16–47). These registers cannot be used for pending the exception numbers 1 to 15. These 15 exceptions have their own support for pending, as explained in **"Exception sources"** on page 58.

7.9 Stack usage for exceptions

When the CPU executes the main code (in thread mode) and an exception request occurs, the CPU stores the state of its general-purpose registers in the stack. It then starts executing the corresponding exception handler (in handler mode). The CPU pushes the contents of the eight 32-bit internal registers into the stack. These registers are the Program and Status Register (PSR), ReturnAddress, Link Register (LR or R14), R12, R3, R2, R1, and R0. Cortex®-M0+ has two stack pointers - MSP and PSP. Only one of the stack pointers can be active at a time. When in thread mode, the Active Stack Pointer bit in the Control register is used to define the current active stack pointer. When in handler mode, the MSP is always used as the stack pointer. The stack pointer in Cortex®-M0+ always grows downwards and points to the address that has the last pushed data.

When the CPU is in thread mode and an exception request comes, the CPU uses the stack pointer defined in the control register to store the general-purpose register contents. After the stack push operations, the CPU enters handler mode to execute the exception handler. When another higher priority exception occurs while executing the current exception, the MSP is used for stack push/pop operations, because the CPU is already in handler mode. See the **"Cortex®-M0+ CPU"** on page 28 for details.

The Cortex®-M0+ uses two techniques, tail chaining and late arrival, to reduce latency in servicing exceptions. These techniques are not visible to the external user and are part of the internal processor architecture. For information on tail chaining and late arrival mechanism, visit the Arm® Infocenter.

7.10 Interrupts and low-power modes

PSoC™ 4 HV PA allows device wakeup from low-power modes when certain peripheral interrupt requests are generated. The Wakeup Interrupt Controller (WIC) block generates a wakeup signal that causes the device to enter Active mode when one or more wakeup sources generate an interrupt signal. After entering Active mode, the ISR of the peripheral interrupt is executed.

The Wait For Interrupt (WFI) instruction, executed by the CM0+ CPU, triggers the transition into Sleep and Deep Sleep modes. The sequence of entering the different low-power modes is detailed in the **"Power modes"** on page 113. Chip low-power modes have two categories of fixed-function interrupt sources:

- Fixed-function interrupt sources that are available only in the Active and Deep Sleep modes (watchdog timer interrupt,)
- Fixed-function interrupt sources that are available only in the Active mode (all other fixed-function interrupts)

7.11 Exceptions – Initialization and configuration

This section covers the different steps involved in initializing and configuring exceptions in PSoC™ 4 HV PA.

1. Configuring the Exception Vector Table Location: The first step in using exceptions is to configure the vector table location as required – either in flash memory or SRAM. This configuration is done by writing bits 31:28 of the VTOR register with the value of the flash or SRAM address at which the vector table will reside. This register write is done as part of device initialization code.
   – It is recommended that the vector table be available in SRAM if the application needs to change the vector addresses dynamically. If the table is located in flash, then a flash write operation is required to modify the vector table contents.
2. Configuring Individual Exceptions: The next step is to configure individual exceptions required in an application.
   – Configure the exception or interrupt source; this includes setting up the interrupt generation conditions. The register configuration depends on the specific exception required.
   – Define the exception handler function and write the address of the function to the exception vector table. **Table 7-1** gives the exception vector table format; the exception handler address should be written to the appropriate exception number entry in the table.
   – Set up the exception priority, as explained in **"Exception priority"** on page 61.
   – Enable the exception, as explained in **"Enabling and disabling interrupts"** on page 62.

7.12 Registers

Table 7-7. List of Registers

Register name	Description
CM0P_ISER	Interrupt Set-Enable Register
CM0P_ICER	Interrupt Clear Enable Register
CM0P_ISPR	Interrupt Set-Pending Register
CM0P_ICPR	Interrupt Clear-Pending Register
CM0P_IPR	Interrupt Priority Registers
CM0P_ICSR	Interrupt Control State Register
CM0P_AIRCR	Application Interrupt and Reset Control Register
CM0P_SCR	System Control Register
CM0P_CCR	Configuration and Control Register
CM0P_SHPR2	System Handler Priority Register 2
CM0P_SHPR3	System Handler Priority Register 3
CM0P_SHCSR	System Handler Control and State Register
CM0P_SYST_CSR	Systick Control and Status Register
CPUSS_CONFIG	CPU Subsystem Configuration Register
CPUSS_SYSREQ	System Request Register

7.13 Associated documents

- Armv6-M Architecture Reference Manual – This document explains the Arm® Cortex®-M0+ architecture, including the instruction set, NVIC architecture, and CPU register descriptions.

8 Flash Memory

The PSoC™ 4 HV PA has a flash module with separate controllers for code flash and data flash. The flash has an accelerator, tightly coupled to the CPU to improve average access times from the flash block. The flash accelerator delivers 85% single-cycle SRAM access performance on average.

The flash program and debug interface are explained in the **"Program and Debug"** on page 360.

8.1 Features

PSoC™ 4 HV PA flash memory has the following features:

- Supports AEC-Q100 Automotive Grade 1 temperature range (Ambient temperature = –40°C to 125°C)
- Up to 128KB of code flash with Error Correction Code (ECC)
- Up to 8KB of data flash with ECC
- 1KB of SFlash available for storing constants with ECC
- Flash macro data: 72-bit width (64-bit data + 8-bit parity)
- ECC parity calculated in hardware
  - Erased flash (all zeros) will not cause ECC errors on read
  - Eight ECC bits allow a single error correction and double error detection
  - ECC error injection
- Implement flash write protect register
- Support for a second flash controller
  - Support for difference sizes for code and data flash
- Flash controller with data buffers to accelerate flash memory accesses
- System Performance Controller Interface (SPCIF) with ECC provides program and erase functionality
- Flash endurance: 100K cycles at 85°C ambient, 10K cycles at 125°C ambient
- Program disturb: 100K cycles at 125°C ambient
- Data retention: Up to 15 years
- Core power supply: V_{CCD} = 1.8 V

8.2 Block diagram

Figure 8-1 gives an overview of the flash controller and SPCIF data path that supports ECC based on a 64-bit word along with 8 bits of ECC parity.

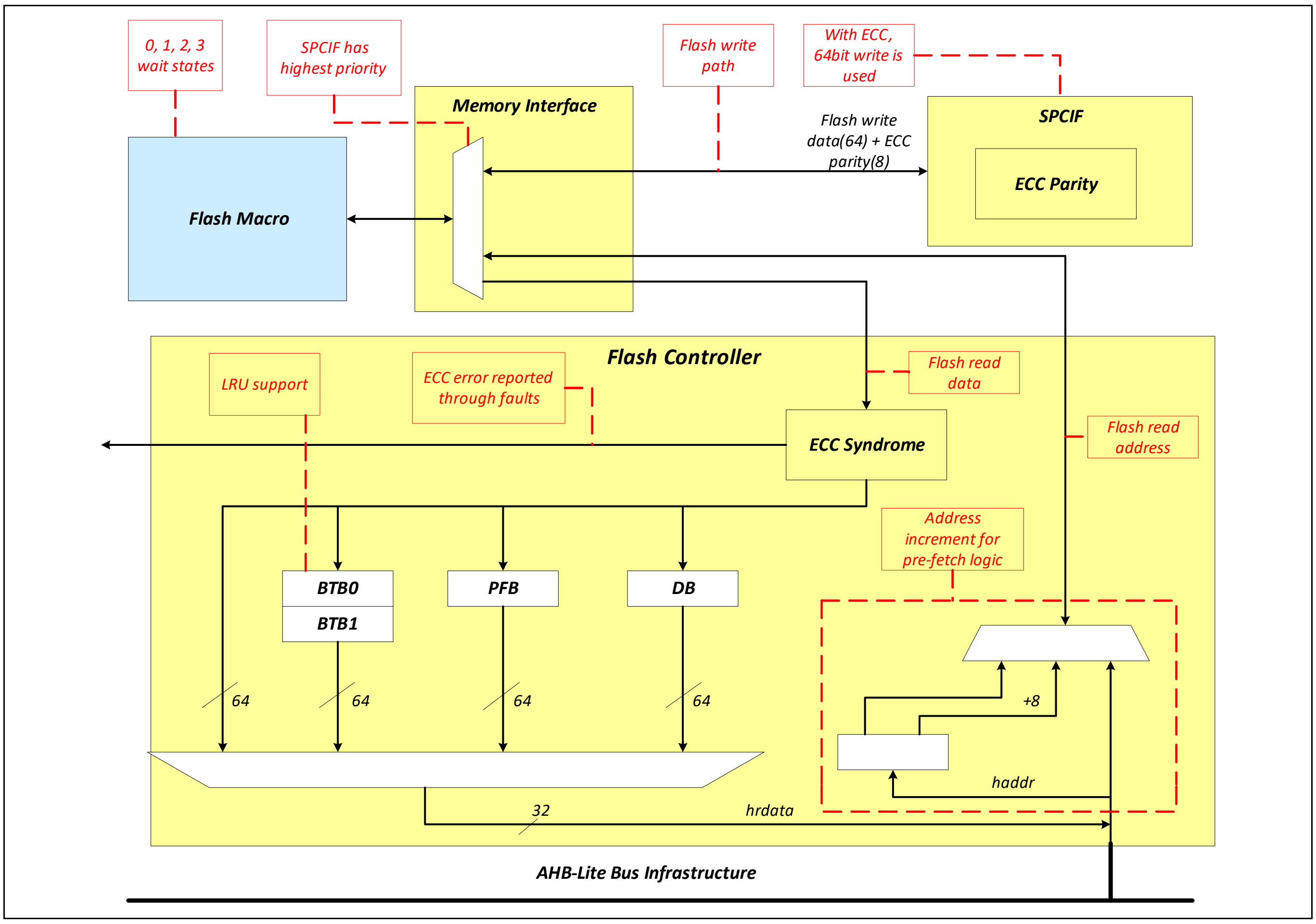

0, 1, 2, 3 wait states
SPCIF has highest priority
Flash write path
With ECC, 64bit write is used
Memory Interface
Flash write data(64) + ECC parity(8)
SPCIF
ECC Parity
Flash Macro
Flash Controller
LRU support
ECC error reported through faults
Flash read data
Flash read address
ECC Syndrome
Address increment for pre-fetch logic
BTB0
BTB1
PFB
DB
64
64
64
64
+8
haddr
32
hrdata
AHB-Lite Bus Infrastructure

Figure 8-1. Flash controller block diagram

Note that there is a second controller (Controller_1) in PSoC™ 4 HV PA devices, which is the same as **Figure 8-1**. **Table 8-1** shows the address mapping for each controllers.

Table 8-1. Address mapping for first and second controllers

Slave Device	Memory Region and Flash Macro	Notes
Flash memory (Controller 0)	0x0000:0000–0x0000:FFFF (Macro 0) 0x0001:0000–0x0001:FFFF (Macro 1)	128KB: Program code region. You can also place data here. Includes the exception vector table, which starts at address 0.
Supervisory Flash memory (Controller 0)	0x0FFF:E000–0x0FFF:E3FF (Macro 0)	1KB: This memory is used for trimming, wounding, protection mode information.
User Supervisory Flash memory (Controller 0)	0x0FFF:E400–0x0FFF:E7FF (Macro 1)	1KB: This memory is used for user data.
Flash memory (Controller 1)	0x1F00:0000–0X1F00:1FFF (Macro 0)	8KB: Data region. A common usage is implementing a load balanced EEPROM for storing data.

8.3 Flash controller and SPCIF

The flash controller performs only read accesses to the flash memory and implements a 72-bit (64-bit data + 8-bit ECC parity) datapath.

The flash controller includes buffers for flash data (see **Figure 8-1**). Four buffers are distinguished:

- Pre-Fetch Buffer (PFB). This buffer provides flash data for linear CPU code accesses.
- Data Buffer (DB). This buffer provides flash data for data accesses (from the CPU and the DW/DMA controller).
- Branch Target Buffer 0 (BTB0). This buffer provides flash data for branch/jump target CPU code accesses.
- Branch Target Buffer 1 (BTB1). This buffer provides flash data for recently used branch/jump target CPU code accesses.

Note that the flash controller buffers store only 64-bit data and have a depth of 1. They do not store 8-bit ECC parity.

The ECC functionality provides:

- Correction of single-bit errors in a memory data word.
- Detection of single- and double-bit errors in a memory data word

The SPCIF provides program and erase functionality for the flash macro.

The flash macro supports a 72-bit width (64-bit data + 8-bit parity) datapath.

8.3.1 System performance controller interface (SPCIF)

The SPCIF provides interface to the on-chip flash memory. SPCIF features include:

- Program and erase functionality for flash memory.
- A timer triggered interrupt signal, which allows the CPU to enter the Sleep power mode or execute code from SRAM during long program/erase operations.

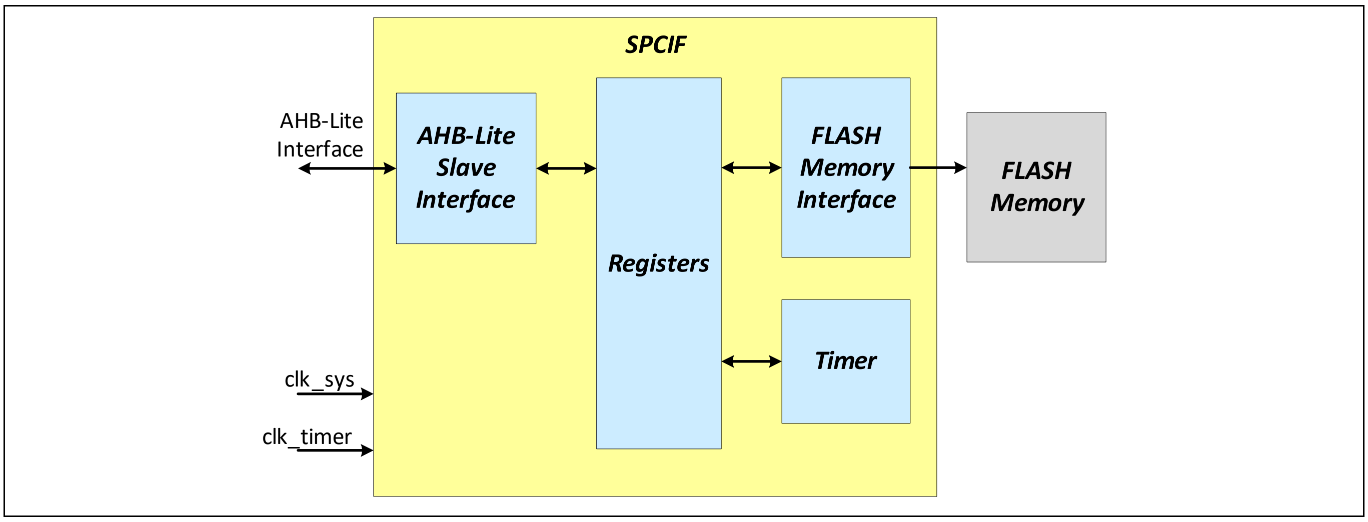


Figure 8-2. SPCIF block diagram

The SPCIF implements a 32-bit timer. The timer is configured to run on a clock, which can be either synchronous or asynchronous to clk_sys (SYSCLK). The timer is used to generate the necessary high-voltage program/erase pulses for the flash memory. The SPCIF can be configured to directly drive the Pump Enable (PE) signal to the High Voltage (HV) pumps to achieve the correct HV pulse as defined by the timer period. For more information on SPCIF registers, see the PSoC™ 4 HV PA Registers TRM.

8.4 ECC implementation

ECC protection is included to the flash for functional safety. The ECC implements a Single Error Correction, Dual Error Detection (SECDED) scheme. In the flash controller, 64 bits of data are covered by eight ECC bits. Note that the PSoC™ 4 HV PA device does not support a disable of ECC functionality for Flash memory.

Single-bit error correction is done in-line, without the need to stall the data returning to the flash controller. **Figure 8-3** shows an overview of the data path of the flash with ECC.

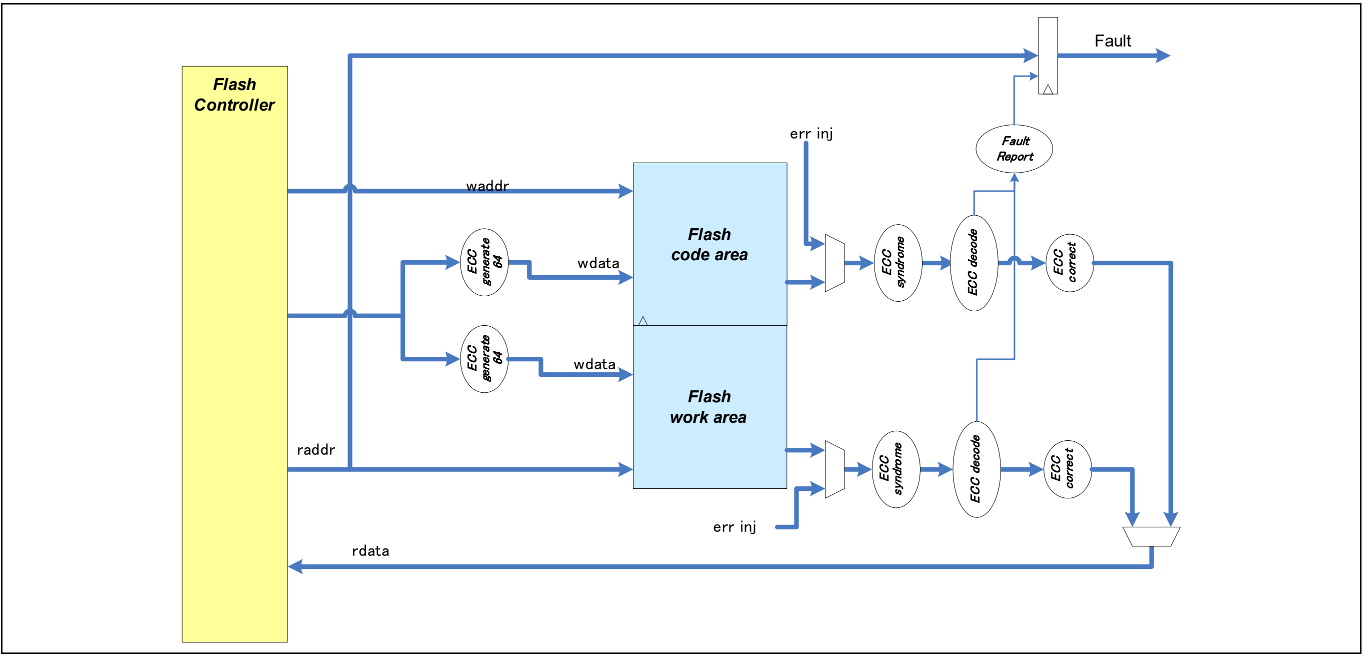


Figure 8-3. Data Path overview for Flash ECC

Programming Code Flash

For a 64-bit program:

- The first data input is stored in a flash macro page latch and not written to flash.
- When the last data input arrives, ECC is calculated, including the buffered data; 64-bit data + 8-bit ECC are programmed to flash.

ECC (Single-Bit Errors)

When the ECC logic detects a single-bit error, the error bit can be corrected. The error correction is inline, without the need to stall the data from returning to the flash controller. This is because the flash read delay is long and the corrected data will be returned with no significant delay. The fault is reported through the regular fault structure. ECC syndrome logic is used to correct the recoverable single-bit errors.

ECC Uncorrectable Errors

If the ECC logic detects that the data has more than one bit wrong then the data cannot be corrected. In this case, the flash returns zero data to the bus master, and returns a bus error response.

Bus Error status

The flash controller reports bus error causes for flash bus transactions.

Fault reporting

Both the correctable and non-correctable ECC errors are reported to the central fault structure in the same way.

All data correction and recovery are left to the ISR. There is no hardware support for writing corrected data back to flash (See the **"Fault subsystem"** on page 87 for more information).

8.4.1 ECC error injection

The ECC error injection functionality is enabled or disabled using FLASH_ECC_INJ_EN bit of CPUSS_FLASH_CTL / CPUSS_FLASHC1_CTL. When “1”, the parity (PARITY[7:0]) is used for a load from the FLASH_WORD_ADDR[23:0] word address.

Table 8-2. Flash ECC Error Injection Control Registers

Register	Bit Field and Bit Name	Description
CPUSS_FLASH_CTL / CPUSS_FLASHC1_CTL	FLASH_ECC_INJ_EN	Enable error injection for Flash main interface. When “1”, the parity (ECC_CTL.PARITY[7:0]) is used for a load from the ECC_CTL.WORD_ADDR[23:0] word address.
CPUSS_FLASHC_ECC_CTL / CPUSS_FLASHC1_ECC_CTL	PARITY	ECC parity to use for ECC error injection at address WORD_ADDR. - For Flash interface ECC, the 8-bit parity PARITY[7:0] is for a 64-bit word.
CPUSS_FLASHC_ECC_CTL / CPUSS_FLASHC1_ECC_CTL	FLASH_WORD_ADDR	Specifies the word address where an error will be injected. - For Flash interface ECC, the word address WORD_ADDR[23:0] is device address A[26:3]. On a Flash main interface read and when FLASH_CTL.MAIN_ECC_INJ_EN bit is “1”, the parity (PARITY[7:0]) replaces the Flash macro parity (Flash main interface read path is manipulated).

The ECC error injection is implemented in the read path for flash memories. The ECC error injection is not implemented in the write path due the reasons listed below:

- Writing to flash memory is a time-consuming task.
- The location at which the ECC error is injected remains corrupted until the next “erase” operation is performed.

When error injection is enabled, the read address is compared to the device address. If they are equal, only the 8-bit parity is over-written.

8.4.2 ECC parity generation by software

This section describes an algorithm to generate the correct ECC parity value with software. Note that this algorithm is not implemented in the hardware. Because the actual algorithm is optimized for hardware performance, it is different from the software algorithm described in this section.

“Value” in this algorithm represents the flash 64-bit data value.

```
CODEWORD_SW[63:0] = {64 {1'b0}};
CODEWORD_SW[63:0] = Value;

ECC_P0_SW = 0x4484_4A88_952A_AD5B;
ECC_P1_SW = 0x1108_9311_26B3_366D;
ECC_P2_SW = 0x0611_1C22_38C3_C78E;
ECC_P3_SW = 0x9821_E043_C0FC_07F0;
ECC_P4_SW = 0xE03E_007C_00FF_F800;
ECC_P5_SW = 0xFFC0_007F_FF00_0000;
ECC_P6_SW = 0xFFFF_FF80_0000_0000;
ECC_P7_SW = 0xD442_2584_4BA6_5CB7;

parity[0] = ^ (CODEWORD_SW & ECC_P0_SW);
parity[1] = ^ (CODEWORD_SW & ECC_P1_SW);
...
parity[7] = ^ (CODEWORD_SW & ECC_P7_SW);

parity[7:0] gives eight bits parity for flash 64-bit data value.
```

Note: “^” means reduction XOR. For example, ^(4’b0011) = 0^0^1^1.

8.4.3 8-bit ECC syndrome logic

The ECC syndrome logic detects and corrects single-bit ECC errors. This logic can be deployed system wide wherever data protected by an 8-bit ECC parity needs to be read, checked for errors, and corrected.

The syndrome is encoded as follows (see **Table 10-16** Fault Reporting Assignments for more details):

- syndrome[7] is ‘0’ and syndrome [6:0] is “0”: no error is detected
  - “cpuss.fault_flashcx_c_ecc” = ‘0’, “cpuss.fault_flashcx_nc_ecc” = ‘0’
- syndrome [7] is ‘0’ and syndrome [6:0] is not “0”: a double error is detected.
  - “cpuss.fault_flashcx_c_ecc” = ‘0’, “cpuss.fault_flashcx_nc_ecc” = “1”,
- syndrome[7] is ‘1’: a single error is detected and syndrome[6:0] specifies the bit error location: (syndrome[6:0] - 1) is the bit error location within CW[127:0] (See CW code below).
  - “cpuss.fault_flashcx_c_ecc” = “1”, “cpuss.fault_flashcx_nc_ecc” = ‘0’.
- CW code

```
W[63:0] = ACTUALWORD[63:0];
P[7:0] = parity[7:0];
A[55:0] = ADDRESS[ADDR_WIDTH-1:0]
CW[127:0] =
            {   P[7], A[55], A[54], A[53], A[33], A[52], A[30], A[29],
             W[63], A[51], A[35], A[40], A[1], A[44], A[8], W[62],
             W[61], A[50], A[42], A[47], A[22], A[38], W[60], A[3],
             W[59], A[28], A[0], W[58], W[57], A[19], W[56], W[55],
             W[54], A[49], A[46], A[43], A[10], A[37], A[5], A[17],
             W[53], A[32], A[23], A[7], W[52], A[12], W[51], W[50],
             W[49], A[36], A[16], A[14], W[48], A[26], W[47], W[46],
             W[45], A[20], W[44], W[43], W[42], W[41], W[40], W[39],
                P[6], A[48], A[39], A[34], A[18], A[45], A[24], A[21],
             W[38], A[41], A[15], A[25], W[37], A[13], W[36], W[35],
             W[34], A[31], A[11], A[9], W[33], A[6], W[32], W[31],
             W[30], A[4], W[29], W[28], W[27], W[26], W[25], W[24],
                P[5], A[27], W[23], A[2], W[22], W[21], W[20], W[19],
             W[18], W[17],W[16], W[15], W[14], W[13], W[12], W[11],
                P[4], W[10], W[9], W[8], W[7], W[6], W[5], W[4],
                P[3], W[3], W[2], W[1], P[2], W[0], P[1], P[0] };
```

8.4.4 Bus error status

The flash controller generates a bus error under the following conditions:

- An occurrence of a Flash macro interface internal error (INTERNAL_ERROR)
- An Uncorrectable Read error occurred when accessing a Flash macro (FLASH_UNCORRECTABLE)
- An access to a non-existent flash address occurred (FLASH_MEMORY_HOLE)
- A protection violation occurred in flash controller (FLASH_PROT_VIO)

Non-correctable errors will also be reported back to the bus master as a bus error. The bus error back to the CPU itself can be disabled by register control. If FLASH_ERR_SILENT = 1, then a bus transfer will not generate a bus error back to the CPU.

Table 8-3. Flash Error Silent Register

Register	Bit Field and Bit Name	Description
CPUSS_FLASH_CTL / CPUSS_FLASHC1_CTL	FLASH_ERR_SILENT	Specifies bus transfer behavior for a non-recoverable error on the Flash macro main interface (either a non-correctable ECC error, a Flash macro main interface internal error, a Flash macro main interface memory hole access, a protection violation error): 0: Bus transfer has a bus error. 1: Bus transfer does not have a bus error; the error is "silent"

Table 8-4. Flash Bus Error Status Registers

Register	Bit Field and Bit Name	Description
CPUSS_FLASHC_BERR_STATUS / CPUSS_FLASHC1_BERR_STATUS	INTERNAL_ERROR	Specifies/registers the occurrence of a Flash macro interface internal error (typically the result of a read access while a program erase operation is ongoing) as a result of a CM0+ access. SW clears this field to "0". HW sets this field to "1" on a Flash macro interface internal error. Typically, SW reads this field after a code section to detect the occurrence of an error.
CPUSS_FLASHC_BERR_STATUS / CPUSS_FLASHC1_BERR_STATUS	FLASH_UNCORRECTABLE	An Uncorrectable Read error occurred when accessing a Flash macro. SW clears this field to "0". HW sets this field to "1" on a Flash macro uncorrectable error. Typically, SW reads this field after a code section to detect the occurrence of an error.

Table 8-4. Flash Bus Error Status Registers (continued)

Register	Bit Field and Bit Name	Description
CPUSS_FLASHC_BERR_STATUS / CPUSS_FLASHC1_BERR_STATUS	FLASH_MEMORY_HOLE	An access to a non-existent Flash address occurred. SW clears this field to "0". HW sets this field to "1" on a Flash macro access to a non-existent address. Typically, SW reads this field after a code section to detect the occurrence of an error.
CPUSS_FLASHC_BERR_STATUS / CPUSS_FLASHC1_BERR_STATUS	FLASH_PROT_VIO	A protection violation occurred in Flash controller. SW clears this field to "0". HW sets this field to "1" on a M0S8 protection violation occurred in Flash controller. Typically, SW reads this field after a code section to detect the occurrence of an error.

8.4.5 ECC error reporting

Table 8-5 provides different actions taken based on the type of ECC error and FLASH_ERR_SILENT. These scenarios are also reported to the central fault structure; the details of fault reporting are explained in the **"Fault subsystem"** on page 87.

Table 8-5. Action taken based on different Error Scenarios

FLASH_ERR_SILENT	Detected Error type	Action taken		Comments
		Bus Error	Fault reporting	
0	1-bit	No	Yes	Bus Error is not generated for 1-bit error.
0	2-bit	Yes	Yes	For a 2-bit error, bus error is reported. Along with the appropriate fault.
0	Flash Read while Flash program/erase is busy	Yes	Yes	–
0	Memory hole during Flash read/write	Yes	Yes	–
1	1-bit	No	Yes	Bus error is not generated for 1-bit error.
1	2-bit	No	Yes	Bus error is suppressed as FLASH_ERR_SILENT is '1'.
1	Flash Read while Flash program/erase is busy	No	Yes	
1	Memory hole during Flash read/write	No	Yes	

8.5 Flash wait states

The access time of flash memories with ECC is around 60 ns. Hence, flash read operation cannot be performed in a single clock cycle at 48-MHz system clock frequency. Wait states will be added when reading data from flash memories.

The FLASH_WS bit of the CPUSS_FLASH_CTL/CPUSS_FLASHC1_CTL registers can be used to configure the wait states required for flash read operation depending on the flash memory used and the system clock frequency.

The actual number of cycles for these wait states depends on the system clock frequency (See **Table 8-6** and **Table 8-7**).

Table 8-6. Flash Wait States

Flash Memory type	System Frequency [MHz]	Wait States
Flash with ECC	SYSCLK ≤ 1/4 × FMAX	0
	SYSCLK ≤ 1/2 × FMAX	1
	SYSCLK ≤ 3/4 × FMAX	2
	SYSCLK ≤ FMAX	3

Table 8-7. FLASH Wait States Register

Register	Bit Field and Bit Name	Description
CPUSS_FLASH_CTL / CPUSS_FLASHC1_CTL	FLASH_WS	Amount of ROM wait states: For Flash With ECC. 0: 0 wait states (SYSCLK ≤ 1/4 × FMAX) 1: 1 wait state (SYSCLK ≤ 1/2 × FMAX) 2: 2 wait states (SYSCLK ≤ 3/4 × FMAX) 3: 3 wait states (SYSCLK ≤ FMAX) 4–15: Future Use - To be used, if three wait states are not sufficient.

8.6 Implement Flash Write Protect Register

The functionality of Flash Write Protect is supported through the following registers in SPCIF.

- SPCIF_FLASH_LOCK
- SPCIF_FLASH_MACRO_WE

SPCIF_FLASH_MACRO_WE.MAC_WRITE_EN controls the Flash macro's write access. This is a bit mask where each bit controls program/erase access to the corresponding macro. When the bit is set to "1", the macro may be programmed/erased; when the bit is set to "0", the macro content is locked.

If the bit transitions from 1 to 0 while the corresponding macro is being erased or programmed, the pumps are immediately disabled where the Flash macro's Pump Enable (pe) is set to low. This may cause the Flash macro memory cells to move to the invalid or unknown state.

Due to this, the software should check for MAC_WRITE_EN before starting any write operation and avoid changing MAC_WRITE_EN during an ongoing write operation. If changing MAC_WRITE_EN during an ongoing write operation is unavoidable, make sure the Flash macro memory cells are not in an unknown state. An erase operation to the same memory location must be issued after MAC_WRITE_EN is changed to unprotected again. This is to ensure the memory cells are transitioned back to a valid state.

- The content of SPCIF_FLASH_MACRO_WE.MAC_WRITE_EN may be changed only when SPCIF_FLASH_MACRO_WE.LOCKED is "0".
- When SPCIF_FLASH_MACRO_WE.LOCKED is "1", hardware will prevent software access to SPCIF_FLASH_MACRO_WE.MAC_WRITE_EN.
- To toggle the value of SPCIF_FLASH_MACRO_WE.LOCKED, a unique 32-bit value of "0xF56B3A81" must be written to SPCIF_FLASH_LOCK.KEY by software.
- The content of SPCIF_FLASH_LOCK.KEY is hidden and not readable by software.

Software can read the toggle of SPCIF_FLASH_MACRO_WE.LOCKED to know the correct key write on this SPCIF_FLASH_LOCK.KEY register.

Table 8-8. Flash Write Protect Registers

Register	Bit Field and Bit Name	Description
SPCIF_FLASH_LOCK	KEY	Write-only register that locks/unlocks access to the FLASH_MACRO_WE register by writing a key value to the register. The key is the 32-bit value 0xF56B3A81. When this bit pattern is written to the register, write access to FLASH_MACRO_WE is toggled between locked and unlocked. The bit FLASH_MACRO_WE.LOCKED toggles upon receipt of the required key value.
SPCIF_FLASH_MACRO_WE	LOCKED	When set, it indicates that write access to FLASH_MACRO_WE.MAC_WRITE_EN is blocked. The value of this bit toggles when the key value is written into FLASH_LOCK.KEY.
SPCIF_FLASH_MACRO_WE	MAC_WRITE_EN	Access control to Flash Macros Write Access register. This is a bit mask where each bit controls program/erase access to the corresponding macro. When the bit is set to '1' the macro may be programmed/erased; when the bit is set to '0', the macro content is locked. The content of this register may only be changed when FLASH_MACRO_WE.LOCKED is '0'.

8.7 Register List

Table 8-9. Flash Memory Register List

Register name	Description
CPUSS	
CPUSS_FLASH_CTL	Flash Control Register
CPUSS_FLASHC1_CTL	Flash Control1 Control Register
CPUSS_FLASHC_BERR_STATUS	Flash Bus Error Status Register
CPUSS_FLASHC1_BERR_STATUS	Flash 1 Bus Error Status Register
CPUSS_FLASHC_ECC_CTL	Flash ECC Control Register
CPUSS_FLASHC1_ECC_CTL	Flash 1 ECC Control Register
SPCIF	
SPCIF_GEOMETRY	Flash/NVL Geometry Information Register
SPCIF_FLASH_LOCK	Flash Lock Register
SPCIF_FLASH_MACRO_WE	Flash Macro Write Enable Register
SPCIF_INTR	SPCIF Interrupt Request Register
SPCIF_INTR_SET	SPCIF Interrupt Set Request Register
SPCIF_INTR_MASK	SPCIF Interrupt Mask Register
SPCIF_INTR_MASKED	SPCIF Interrupt Masked Request Register
SPCIF1_GEOMETRY	Flash/NVL Geometry Information Register
SPCIF1_FLASH_LOCK	Flash Lock Register
SPCIF1_FLASH_MACRO_WE	Flash Macro Write Enable Register
SPCIF1_INTR	SPCIF1 Interrupt Request Register
SPCIF1_INTR_SET	SPCIF1 Interrupt Set Request Register
SPCIF1_INTR_MASK	SPCIF1 Interrupt Mask Register
SPCIF1_INTR_MASKED	SPCIF1 Interrupt Masked Request Register

9 SRAM

The PSoC™ 4 HV PA has a volatile static random access memory (SRAM) with Error Correction Code (ECC). The SRAM is used by the processor for storing variables and can program code, which can be written and executed in SRAM. SRAM memory is retained in all power modes (Active, Sleep, and Deep Sleep). At power-up, SRAM is uninitialized and should be written by application code before reading.

9.1 Features

PSoC™ 4 HV PA SRAM has the following features:

- Up to 8 KB of SRAM with ECC
- SRAM macro data: 40-bit width (32-bit data + 7-bit parity)
- Synchronous read/write
- ECC parity calculated in hardware
  - Seven ECC bits allows a single error correction and double error detection
  - ECC error injection
- Separate power supplies for core and periphery

Note that there is a redundancy of 1-bit on MSB (sram_data_in/out[39]) in SRAM. This redundant bit is ignored during the read cycle and does not participate in ECC. For the completion of 40-bits, fixed value ‘1’b0’ is written to this redundant bit.

9.2 Block diagram

Figure 9-4 gives an overview of the SRAM controller block diagram with ECC support.

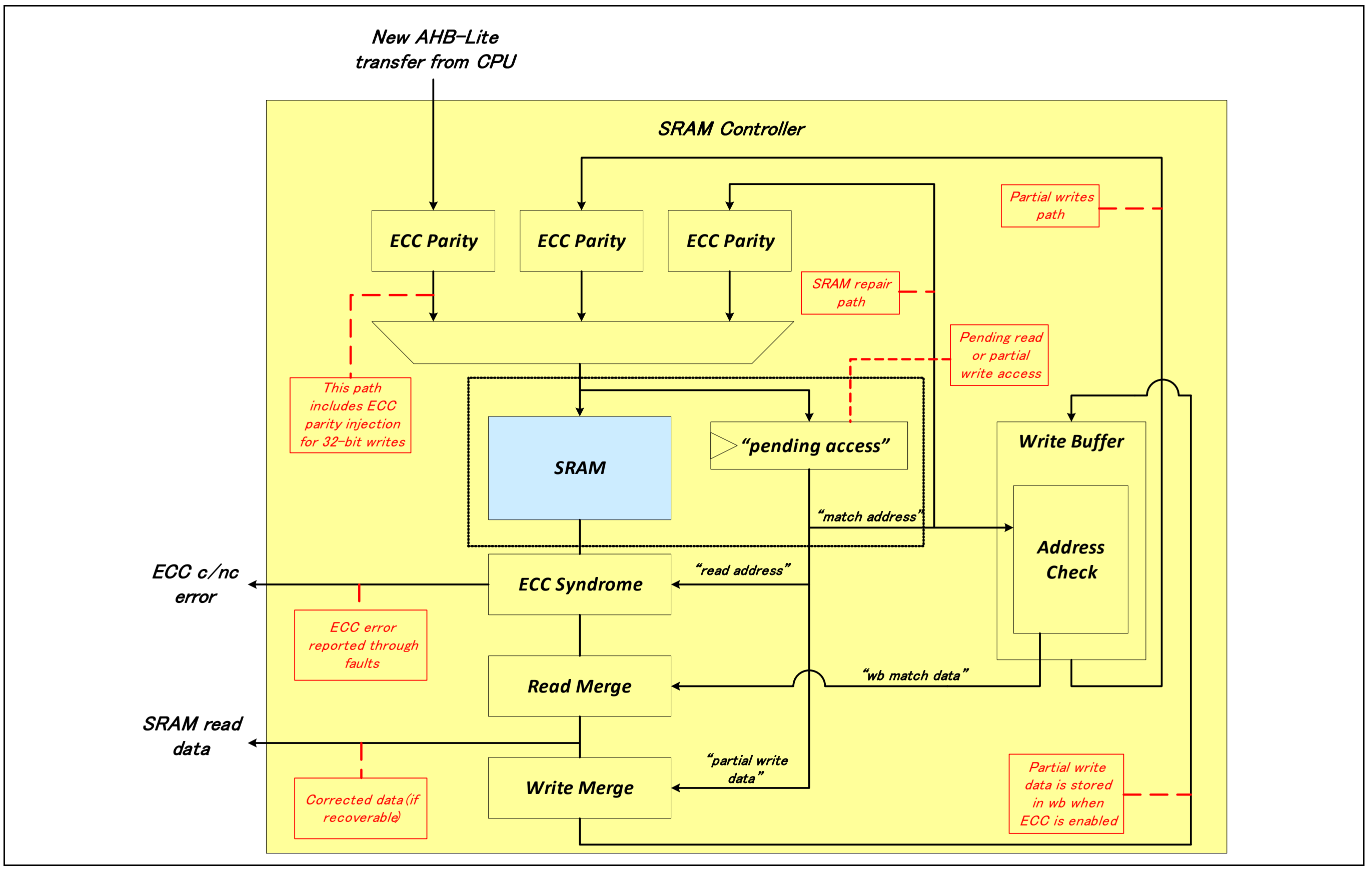


Figure 9-4. SRAM Controller block diagram

9.3 SRAM read/write accesses with ECC functionality

SRAM accesses originate from one of the following paths:

- AHB-Lite transfers.
- Write buffer requests (only when CPUSS_RAM_CTL.ECC_ENABLE is '1').
- SRAM repair requests (only when CPUSS_RAM_CTL.ECC_ENABLE is '1').

Each path has dedicated ECC parity logic. The paths are evaluated in the following priority (from high to low priority):

- SRAM repair requests
- Write buffer requests, when the write buffer is full
- AHB-Lite requests
- Write buffer requests, when the write buffer is not empty

The AHB-Lite transfers are the origin for all SRAM accesses; that is, the write buffer and SRAM repair requests result from AHB-Lite transfers. The SRAM controller differentiates between the following three types of AHB-Lite transfers:

- AHB-Lite read transfers
- 32-bit AHB-Lite write transfers
- 8-bit and 16-bit AHB-Lite write transfers (partial AHB-Lite write transfers)

Table 9-10. AHB-Lite transfers

AHB-Lite Access	Action taken
AHB-Lite read transfers with ECC	• If address matches in write buffer, and valid = 1, AHB data is read from write buffer and data is not read from SRAM. • If address does not match in write buffer, OR valid = 0; AHB data is read from SRAM.
32-bit write access	• If address matches in write buffer, and valid = 1; write buffer is invalidated and AHB data is directly written to SRAM • If address does not match in write buffer, OR valid = 0; AHB data is directly written to SRAM
Partial AHB-Lite write transfers with ECC	• If address matches in write buffer, and valid = 1, AHB data is merged with data read from write buffer and data is not read from SRAM. Merged data is written back to write buffer. • If address does not match in write buffer, OR valid = 0; AHB data is read from SRAM and merged data is written back to write buffer.

9.3.1 32-bit AHB-Lite write transfers with ECC

A 32-bit AHB-Lite write transfer is translated into an SRAM write access. The ECC parity logic is used. If the write address matches in the write buffer, the matching write buffer entries have stale data and these entries are invalidated. The ECC is computed and written to SRAM.

9.3.2 Partial AHB-Lite write transfers with ECC

A partial AHB-Lite write transfer is translated into an SRAM read access and an SRAM write access.

A partial write transfer requires an SRAM read access to retrieve the “missing” data bytes from the SRAM. If the read address matches in the write buffer, the SRAM has stale data and the write buffer provides the requested read data. The requested read data is merged with the partial write data to provide a complete 32-bit data word. The address and the merged write data are written to the write buffer. A future write buffer request results in an SRAM write access with the merged write data. Only the partial AHB-Lite write transfers use the write buffer.

Since partial write transfers involve SRAM read accesses the following ECC error scenarios are possible:

- No ECC Error: No action taken
- Correctable Error (1-bit error):
  - This scenario requires an SRAM update. The corresponding SRAM address needs to be written/repaired with the corrected code word. This additional SRAM write access is performed through the SRAM repair request path. This functionality is only enabled when CPUSS_RAM_CTL.ECC_AUTO_CORRECT is ‘1’.
  - This corrected 32-bit data word is then merged with the write data with appropriate mask and stored in the write buffer. A future write buffer request results in an SRAM write access with the merged write data.
- Uncorrectable Error (2-bit error):
  - Since there are 2-bit errors, no correction is made to the data word read from the SRAM and no merged data is stored in the write buffer.

9.3.3 AHB-Lite read transfers with ECC

An AHB-Lite read transfer is translated into an SRAM read access. The ECC syndrome logic is used, which corrects recoverable (1-bit) errors. If the read address matches in the write buffer, the SRAM has stale data and the write buffer provides the requested read data.

Following are the possible ECC error scenarios with AHB-Lite read transfers:

- No ECC Error: No action taken
- Correctable Error (1-bit error):
  - This scenario requires an SRAM update. The corresponding SRAM address needs to be written/repaired with the corrected code word. This additional SRAM write access is performed through the SRAM repair request path. This functionality is only enabled when CPUSS_RAM_CTL.ECC_AUTO_CORRECT is ‘1’.
- Uncorrectable Error (2-bit error):
  - Since there are 2-bit errors, no correction is made to the data word read from SRAM and no merged data is stored in the write buffer.

9.4 Write buffer functionality

The write buffer component is used only during 8-bit and 16-bit AHB-Lite write transfers (partial AHB-Lite write transfers) with CPUSS_RAM_CTL.ECC_ENABLE enabled.

The buffer allows partial AHB-Lite SRAM write accesses with ECC to be postponed, allowing for more performance critical AHB-Lite requests to "overtake" write buffer requests. This means the a partial AHB-Lite write data is first stored in write buffer and then stored in the SRAM when there are no new or outstanding AHB-Lite transfers.

Note 1: In case of 1-bit ECC error during SRAM read operation, the corrected data is directly written back to SRAM. (Write buffer is not used to hold the corrected data).

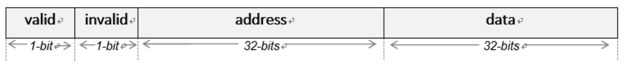
Note 2: The 7-bit parity generated during the partial AHB-Lite SRAM accesses with ECC is not stored in the write buffer.

Memory consistency is guaranteed by matching the SRAM access address with the entries in the write buffer:

- "matching" address during SRAM read access means the SRAM has stale/old data and the data from write buffer should be used.
- "matching" address during SRAM write access means the entry in write buffer needs to be invalidated.

A valid and invalid bit in the write buffer is used to convey the above information.

The write buffer is constructed as a FIFO with four entries, as shown below.



Each entry consists of:

- A valid field.
- An invalidated field.
- A word address.
- A 32-bit data word.

Note 1: A FIFO structure is one where the order in which entries are written is the same as the order in which entries are read.

Note 2: The merged write data that is written to the write buffer is always a 32-bit data word. Therefore, no byte mask is required.

The following actions are taken when the write buffer is accessed:

- When an entry is added to write buffer:
  - Valid bit is set "1"
  - Invalid bit is set "0"
  - Address and data are stored in respective fields.
- When an entry is read from write buffer:
  - If valid bit is "0" (invalid bit is "1"), the write buffer data is ignored, and no SRAM access is performed.
  - If valid bit is "1", the write buffer data is transferred to SRAM at the location mentioned in write buffer address field. After the transfer is complete the valid bit is set to "0".

The state of the write buffer is reflected by CPUSS_RAM_STATUS.WB_EMPTY, which determines the priority of when a valid write buffer data can be transferred to SRAM. This priority decision is elaborated in **"SRAM read/write accesses with ECC functionality"** on page 80.

The write buffer is not retained in Deep Sleep power mode. Therefore, when transitioning to system Deep Sleep power mode, the write buffer should be empty. This requirement is typically met, as a transition to Deep Sleep power mode also requires that there are no outstanding AHB-Lite transfers.

9.5 ECC implementation

The ECC engine provides supports for Hamming code with an additional parity bit. This code supports single error correction, double error detection (SECDED). The ECC is applied to the SRAM data and the SRAM address.

- The ECC corrects single bit errors in an SRAM data (stored in SRAM memory).
- The ECC detects single and double bit errors in an SRAM data (stored in SRAM memory) and the SRAM address logic.

As mentioned earlier the SRAM stores 40-bit of data: a 32-bit data word, a 7-bit parity, and redundant 1-bit. The 32-bit SRAM data word allows 8-bit, 16-bit and 32-bit AHB-Lite bus transfers in a single cycle. The 7-bit parity provides error correction and error detection.

A 7-bit SECDED parity covers up to 57 data bits, which also provides space for an additional 57 – 32 = 25 data bits. These 25 additional data bits will be used for the SRAM address.

9.5.1 ECC enable and disable

The ECC functionality can be disabled or enabled using CPUSS_RAM_CTL.ECC_ENABLE. When ECC functionality is enabled following components are used to implement the desired ECC functionality:

- ECC Parity
- ECC Syndrome
- Write Buffer

Note: With ECC functionality, SRAM read/write data path is changed from 32-bit to 39-bits (including 7 bits of ECC).

9.5.2 ECC error injection

The ECC error injection functionality is enabled or disabled using CPUSS_RAM_CTL.ECC_INJ_EN.

The ECC error injection is implemented in the write path for the SRAM memories. The ECC error injection is implemented only for 32-bit AHB write requests. There is no ECC error injection for partial AHB write requests.

When ECC_INJ_EN = 1, the ECC ERROR is injected by writing the corrupted ECC parity to the ECC_TEST.SYND_DATA at the desired location specified by ECC_TEST.WORD_ADDR.

After writing the above registers, since the ECC is injected on the write path, the SW needs to issue an SRAM write to that location. This results in the corrupted parity is being written to the SRAM.

Table 9-11. SRAM ECC Error Injection Registers

Register	Bit Field and Bit Name	Description
CPUSS_RAM_CTL	ECC_INJ_EN	Enable error injection. 0: Syndrome is source from ECC Syndrome hardware. 1: ECC_TEST.SYND_DATA is used when a full 32-bit write is done to the ECC_TEST.WORD_ADDR word address of SRAM.
CPUSS_ECC_TEST	SYND_DATA	ECC syndrome to use for error injection at address WORD_ADDR.
CPUSS_ECC_TEST	WORD_ADDR	Specifies the word address where an error will be injected. On a write transfer to this SRAM address and when the corresponding RAM_CTL.ECC_INJ_EN bit is '1', the SYND_DATA is injected into the corresponding SRAM.

9.5.3 ECC parity generation by software

To inject the ECC error for fault generation, ECC parity must be generated by software. Follow this procedure to generate a 7-bit ECC parity.

```
CODEWORD_SW[63:0] = {64 {1'b0}};
CODEWORD_SW[31:0] = ACTUALWORD[31:0];
ADDR_WIDTH = log2(RAM_SIZE);
CODEWORD_SW[ADDR_WIDTH+29:32] = ADDR[ADDR_WIDTH-1:2];
```

Note: RAM_SIZE is the size of RAMx, where "x" is the RAM unit number.

```
ECC_P0_SW = 0x037F_36DB_2254_2AAB;
ECC_P1_SW = 0x05BD_EB5A_4499_4D35;
ECC_P2_SW = 0x09DD_DCEE_08E2_71C6;
ECC_P3_SW = 0x11EE_BBA9_8F03_81F8;
ECC_P4_SW = 0x21F6_D775_F003_FE00;
ECC_P5_SW = 0x41FB_6DB4_FFFC_0000;
ECC_P6_SW = 0x8103_FFF8_112C_965F;
```

As shown here, reduction XOR of the ANDed result of CODEWORD_SW[63:0] and respective ECC constants will give a single parity bit.

```
parity[0] = ^ (CODEWORD_SW[63:0] & ECC_P0_SW);
parity[1] = ^ (CODEWORD_SW[63:0] & ECC_P1_SW);
…
parity[6] = ^ (CODEWORD_SW[63:0] & ECC_P6_SW);
parity[6:0] gives seven bits parity for 32 bits ACTUALWORD[31:0].
```

Note: "^" means reduction XOR. For example, ^(4'b0011) = 0^0^1^1.

9.5.4 7-bit ECC syndrome logic

The ECC syndrome logic detects and corrects single-bit ECC errors. This logic can be deployed system wide wherever data protected by a 7-bit ECC parity needs to be read, checked for errors, and corrected.

The syndrome is encoded as follows (see **Table 10-16** Fault Reporting Assignments for more details):

- syndrome[6] is '0' and syndrome [5:0] is "0": no error is detected
  - "cpuss.fault_ramc_c_ecc" = '0', "cpuss.fault_ramc_nc_ecc" = '0'
- syndrome [6] is '0' and syndrome [5:0] is not "0": a double error is detected.
  - "cpuss.fault_ramc_c_ecc" = '0', "cpuss.fault_ramc_nc_ecc" = '1',
- syndrome[6] is '1': a single error is detected and syndrome[5:0] specifies the bit error location: (syndrome[5:0] - 1) is the bit error location within CW[63:0] (See CW code below).
  - "cpuss.fault_ramc_c_ecc" = '1', "cpuss.fault_ramc_nc_ecc" = '0'.
- CW code

```
W[31:0] = ACTUALWORD[31:0];
P[6:0] = parity[6:0];
A[24:0] = ADDRESS[ADDR_WIDTH-1:0];
CW[63:0] = { P[ 6], A[24], A[23], A[22], A[ 5], A[21], A[8], A[17],
                   W[31], A[20], A[14], A[10], A[ 2], A[ 4], W[30], W[29],
                   W[28], A[19], A[11], A[ 7], W[27], A[13], W[26], W[25],
                   W[24], A[16], W[23], W[22], W[21], W[20], W[19], W[18],
                     P[ 5], A[18], A[15], A[12], W[17], A[ 9], W[16], A[ 0],
                   W[15], A[ 6], W[14], W[13], W[12], W[11], W[10], W[ 9],
                     P[ 4], A[ 3], W[ 8], W[ 7], W[ 6], W[ 5], W[ 4], W[ 3],
                     P[ 3], A[ 1], W[ 2], W[ 1], P[ 2], W[ 0], P[ 1], P[ 0] };
```

9.5.5 ECC error reporting status

The SRAM controller indicates a correctable/uncorrectable error in ECC_CAPTURE_ADDR_SYNDROME_VALID bit of CPUSS_RAM_ECC_STATUS0 register.

If either a correctable or non-correctable error occurs, ECC_CAPTURE_ADDR_SYNDROME_VALID bit will be set. This register should be cleared by FW. ECC_CAPTURE_ADDR_31_2 and ECC_CAPTURE_SYNDROME registers will store the first such occurrence of the address and syndrome causing the ECC error.

If the error occur, ECC_CAPTURE_ADDR_31_2 will store the CPUSS address that caused the first correctable/uncorrectable error interrupt. And ECC_CAPTURE_SYNDROME will store the CPUSS Syndrome that caused the first error interrupt.

Table 9-12. SRAM ECC Status Registers

Register	Bit Field and Bit Name	Description
CPUSS_RAM_ECC_STATUS0	ECC_CAPTURE_ADDR_SYNDROME_VALID	ECC capture address and syndrome valid. 0: ECC_CAPTURE_ADDR_31_2 not valid 1: ECC_CAPTURE_ADDR_31_2 valid - SW writes a 1 to clear
CPUSS_RAM_ECC_STATUS0	ECC_CAPTURE_ADDR_31_2	Snapshot of the CPUSS address that caused the first correctable/uncorrectable error interrupt. When first correctable/uncorrectable error interrupt is generated, ECC_CAPTURE_ADDR_SYNDROME_VALID is transitioned from 0 to 1, and this register captures the first error address. The register only captures bits 31–2 of the address.
CPUSS_RAM_ECC_STATUS1	ECC_CAPTURE_SYNDROME	Snapshot of the CPUSS syndrome that caused the first correctable/uncorrectable error interrupt. When first correctable/uncorrectable error interrupt is generated, ECC_CAPTURE_ADDR_SYNDROME_VALID is transitioned from 0 to 1, and this register captures the first Syndrome corresponding to the Error.

9.5.6 ECC error reporting

Table 9-13 provides the different actions taken based type of ECC error for Partial AHB-Lite write transfers, and **Table 9-14** provides in case of AHB-Lite read transfers. The details of fault reporting are explained in the **“Fault subsystem”** on page 87.

Table 9-13. Error Scenarios for Partial AHB-Lite write transfers

Detected Error type	Action taken		Comments
	Bus Error	Fault reporting	
1-bit	No	Yes	Bus error is not generated for 1-bit error.
2-bit	No	Yes	In case of 2-bit error of write transfers, bus error is not reported.

Table 9-14. Error Scenarios for AHB-Lite read transfers

Detected Error type	Action taken		Comments
	Bus Error	Fault reporting	
1-bit	No	Yes	Bus error is not generated for 1-bit error.
2-bit	Yes	Yes	In case of 2-bit error of read transfers, bus error is reported. Along with an appropriate fault.

9.6 Register list

Table 9-15. SRAM Memory Register list

Register name	Description
CPUSS_RAM_CTL	RAM Control Register
CPUSS_RAM_STATUS	RAM Controller 0 Status Register
CPUSS_RAM_ECC_STATUS0	RAM ECC Status 0 Register
CPUSS_RAM_ECC_STATUS1	RAM ECC Status 1 Register
CPUSS_ECC_TEST	ECC Test Register

10 Fault subsystem

The fault subsystem contains information about faults that occur in the system. The subsystem can cause a reset, give a pulse indication, or trigger another peripheral. The PSoC™ 4 HV PA uses a centralized fault report structure. The centralized nature allows for a system-wide, consistent handling of faults, which simplifies software development as follows:

- Only a single fault interrupt handler is required
- The fault report structure provides the fault source and additional fault-specific information from a single set of Memory Mapped Input/Output (MMIO) registers; that is, no iterative search is required for the fault source and fault information
- All pending faults are available from a single set of MMIO registers

The fault subsystem captures faults related to:

- SRAM controller ECC errors
- FLASH controller ECC errors
- CRWDT fault output
- HVREG fault output

Note that some of the above faults also result in errors on the bus infrastructure.

These faults are communicated as a bus error to the master of the faulting bus transfer. The fault subsystem only captures faults. It does not take any action to correct it.

10.1 Fault report structure

Figure 10-5 gives an overview of the fault report structure.

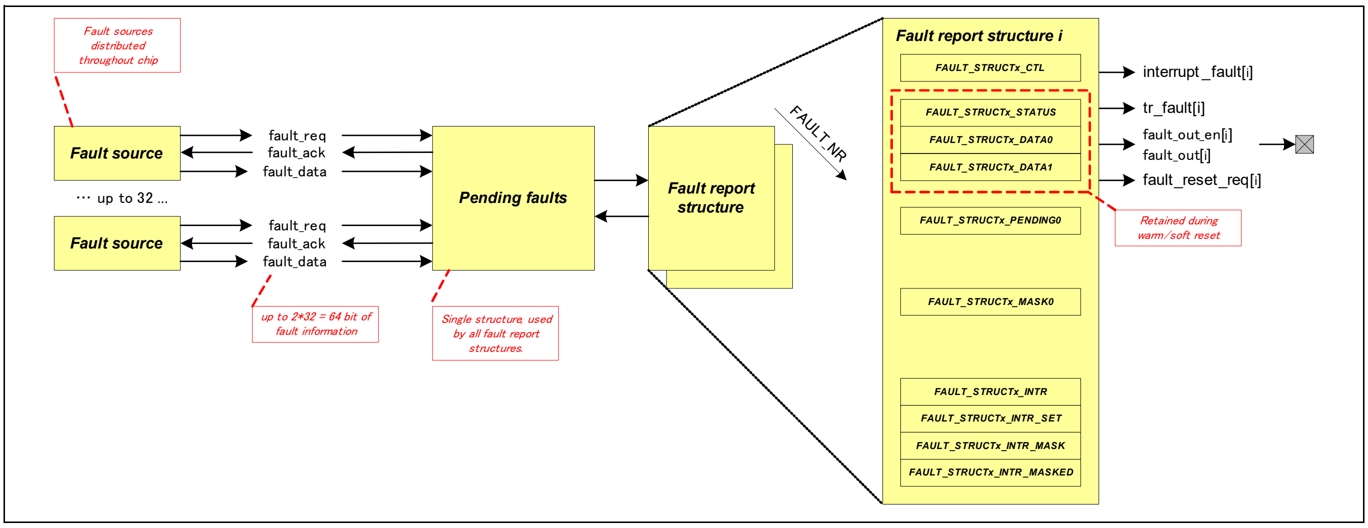


Figure 10-5. Fault reporting structure

The number of faults reporting structures (FAULT_NR) is up to 2. Each structure has a dedicated set of control and status registers, and captures a single fault. The captured fault information includes:

- A validity bit field that indicates a fault is captured (FAULT_STRUCTx_STATUS.VALID).
- A fault index that identifies the fault source (FAULT_STRUCTx_STATUS.IDX).
- Additional fault information describing fault specifics (FAULT_STRUCTx_DATA0 and 1). This additional information is fault type-specific. Most fault types use only a few of the FAULT_STRUCTx_DATA0 and 1 registers.

In addition to the captured fault information, each fault report structure supports a signaling interface to notify the rest of the system of the captured fault. This interface supports the following:

- A fault interrupt (interrupts_fault[i]). This interrupt is supported by the platform interrupt registers: FAULT_STRUCTx_INTR, FAULT_STRUCTx_INTR_SET, FAULT_STRUCTx_INTR_MASK, and FAULT_STRUCTx_INTR_MASKED. Only a single interrupt cause is present: FAULT (indicating that a fault is detected). The FAULT_STRUCTx_INTR_MASK register provides a mask/enable for the cause. The interrupt cause is set to ‘1’ when a fault is captured.
- A trigger (tr_fault[i]). An enabled trigger is activated (generating a two-cycle ‘1’ pulse) when FAULT_STRUCTx_STATUS.VALID is set to ‘1’. The trigger is enabled by FAULT_STRUCTx_CTL.TR_EN. The trigger can be connected to a DMA controller, for example, which can transfer captured fault information from the fault report structure to memory and can clear the FAULT_STRUCTx_STATUS.VALID field. For failure analysis, a memory location that is retained during warm/soft reset is desirable.
- An output signal (fault_out[i]). An enabled output signal is active ‘1’ when FAULT_STRUCTx_STATUS.VALID is ‘1’. The output signal is enabled by FAULT_STRUCTx_CTL.OUT_EN. It can be used to communicate non-recoverable faults, for example, to off-chip components (possibly resulting in a device reset).
- A fault reset request (fault_reset_req[i]). An enabled request is active ‘1’ when FAULT_STRUCTx_STATUS.VALID is ‘1’. The request is enabled by FAULT_STRUCTx_CTL.RESET_REQ_EN. The reset request feeds into the logic that generates a warm/soft reset.

The four different signaling interfaces provided have their own ‘enable’ functionality. Each enabled interface is activated when FAULT_STRUCTx_STATUS.VALID is ‘1’.

As the system resources subsystem (SRSS) has a single fault_reset_req input signal, the individual fault_reset_req[i] signals are combined (logical OR’d) into a single fault_reset_req signal.

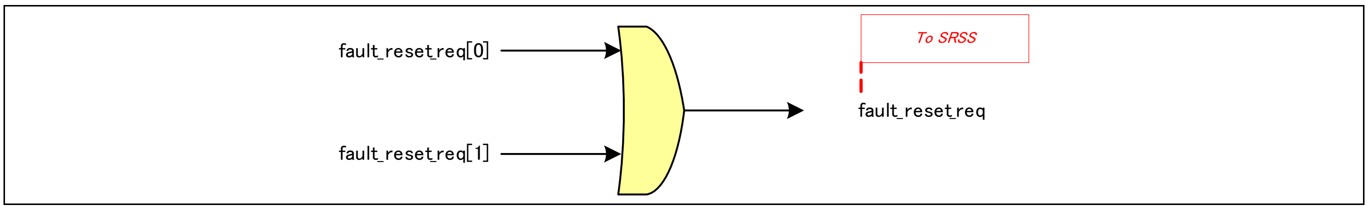


Figure 10-6. Fault reset

A central structure, shared by all fault report structures, keeps track of all pending faults in the system. The FAULT_STRUCTx_PENDING0 registers reflect which of the fault sources are pending. These registers provide a dedicated pending bit for up to 32 fault sources. The FAULT_STRUCTx_PENDING0 registers are mirrored in each of the fault report structures. The fault source numbering scheme follows the numbering scheme of FAULT_STRUCTx_STATUS.IDX.

The fault sources corresponding to a pending bit (which is set) are the ones that are not yet captured by any of the fault structures. When a pending fault is captured by a fault structure, the associated pending bit is cleared to '0'. Each fault report structure is selective in the faults it captures.FAULT_STRUCTx_MASK0 reflect which pending fault source is captured by a fault structure. These faults are referred to as "enabled" faults. The FAULT_STRUCTx_MASK0 registers are unique to each fault structure. This allows for the following:

- One fault report structure is used to capture recoverable faults and another is used to capture non-recoverable faults. The former can be used to generate a fault interrupt and the latter can be used to activate a chip output signal or a reset request.
- Two fault report structures are used to capture the same faults. This first fault is captured by the structure with the lower index (for example, fault structure 0) and the second fault is captured by the structure with the higher index (for example, fault structure 1).
  Note: FAULT_STRUCTx_STATUS.VALID bits are different for each of the fault structures. As an example, consider that the CRWDT lower threshold is linked to Fault Structure#0 and higher threshold is linked to Fault Structure#1.
  Fault Structure#0 occurs first to give a warning; then, Fault Structure#1 occurs to trigger a reset.

A fault structure only captures "enabled" faults when FAULT_STRUCTx_STATUS.VALID is '0'.

When a fault is captured, the hardware sets FAULT_STRUCTx_STATUS.VALID to '1'. In addition, the hardware clears the associated pending bit to '0'. When a fault structure is processed, the software (if the fault is processed by an interrupt handler) or a DMA transfer (if a triggered DMA transfer copied the captured fault information) should clear FAULT_STRUCTx_STATUS.VALID to '0'.

Note that fault capturing does not consider FAULT_STRUCTx_INTR.FAULT:

- Fault capturing is only conditioned by FAULT_STRUCTx_STATUS.VALID being '0'.
- If an interrupt handler is used to process the fault structure, software should clear FAULT_STRUCTx_INTR.FAULT to '0'.

10.2 Fault reporting assignments

The fault subsystem captures faults related to:

- SRAM controller correctable and non-correctable ECC errors
- FLASH controller correctable and non-correctable ECC errors for both FLASH0 and FLASH1
- FLASH0/1 controller bus error related fault output
- CRWDT fault output
- HVREG fault output

The 10 fault sources in the PSoC™ 4 HV PA device are listed in **Table 10-16**.

As mentioned previously, a central structure shared by all fault report structures keeps track of all pending faults in the system (FAULT_STRUCTx_PENDING0). The Fault column in **Table 10-16** provides the "bit mask" of source causing the fault (FAULT_STRUCTx_PENDING0 and FAULT_STRUCTx_MASK0); the Data0/Data1 in Description column indicate FAULT_STRUCTx_DATA0/DATA1.

Table 10-16. Fault reporting assignments

Fault	Source	Description
0	cpuss.fault_ramc_c_ecc	System memory controller 0 correctable ECC violation: DATA0[31:0]: Violating address. DATA1[6:0]: Syndrome of 32-bit SRAM code word.
1	cpuss.fault_ramc_nc_ecc	System memory controller 0 non-correctable ECC violation. DATA0[31:0]: Violating address. DATA1[6:0]: Syndrome of 32-bit SRAM code word.
2–3	Reserved	–
4	cpuss.fault_flashc_c_ecc	Flash controller 0 correctable ECC violation DATA0[31:0]: Violating address DATA1[7:0]: Syndrome of 64-bit FLASH code word.
5	cpuss.fault_flashc_nc_ecc	Flash controller 0 non-correctable ECC violation. DATA0[31:0]: Violating address DATA1[7:0]: Syndrome of 64-bit FLASH code word.
6	cpuss.fault_flashc_bus_err	Flash controller 0 Bus Error. DATA1[31]: Flash Read during Write operation (INTERNAL_ERROR) DATA1[30]: Memory Hole (FLASH_MEMORY_HOLE) DATA1[29]: Protection Violation (FLASH_PROT_VIO) DATA1[9:8]: Master Identifier
7	cpuss.fault_flashc1_c_ecc	Flash controller 1 correctable ECC violation. DATA0[31:0]: Violating address DATA1[7:0]: Syndrome of 64-bit FLASH code word.
8	cpuss.fault_flashc1_nc_ecc	Flash controller 1 non-correctable ECC violation. DATA0[31:0]: Violating address DATA1[7:0]: Syndrome of 64-bit FLASH code word.
9	cpuss.fault_flashc1_bus_err	Flash controller 1 Bus Error DATA1[31]: Flash Read during Write operation (INTERNAL_ERROR) DATA1[30]: Memory Hole (FLASH_MEMORY_HOLE) DATA1[29]: Protection Violation (FLASH_PROT_VIO) DATA1[9:8]: Master Identifier
10	srss.fault_crwdt	Fault output for CRWDT DATA0[0]: CRWDT LOWER_LIMIT DATA0[1]: CRWDT UPPER_LIMIT DATA0[2]: CRWDT WARN_LIMIT
11	hvss.fault_pwr	Fault output for HVREG DATA[0]: HVREG power not good
12–31	Reserved	–

10.3 Fault and reset

As mentioned, a captured fault may result in a warm/soft reset. This type of reset brings regular MMIO registers to their default/reset state. This is not acceptable for the registers that capture fault information; for failure analysis, fault information should be retained during a warm/soft reset. Therefore, the FAULT_STRUCTx_STATUS and FAULT_STRUCTx_DATA0 and 1 registers are connected to a cold reset. This illustrates another benefit of centralized fault report structures: only the centralized structure is connected to a cold reset. The multiple fault sources that are scattered throughout the system can use the regular reset, as a copy of the fault information is captured by the fault structure.

Note: When the fault is configured to trigger reset, then debugging of the configured fault structure is not possible.

10.4 Fault and power modes

The fault report structure functionality is available only in Active/Sleep power modes (it is an Active functionality):

- Deep Sleep fault sources are not supported. These fault sources require dedicated solutions.
- The interfaces between the active fault sources and the centralized fault report structures is reset in Deep Sleep power mode. Note that the fault information is retained.

As the fault report structure is an active functionality, pending faults (in the FAULT_STRUCTx_PENDING0 registers) are not retained when transitioning to Deep Sleep power mode. This is acceptable, because the fault source itself is an active functionality.

10.5 Register list

Table 10-17. Fault subsystem register list

Register name	Description
FAULT_STRUCTx_CTL	Fault Control Register
FAULT_STRUCTx_STATUS	Fault Status Register
FAULT_STRUCTx_DATA0	Fault Data 0 Register
FAULT_STRUCTx_DATA1	Fault Data 1 Register
FAULT_STRUCTx_PENDING0	Fault Pending 0 Register
FAULT_STRUCTx_MASK0	Fault Mask 0 Register
FAULT_STRUCTx_INTR	Interrupt Register
FAULT_STRUCTx_INTR_SET	Interrupt Set Register
FAULT_STRUCTx_INTR_MASK	Interrupt Mask Register
FAULT_STRUCTx_INTR_MASKED	Interrupt Masked Register

Section B: System Resources Subsystem (SRSS)

This section encompasses the following chapters:

Top Level Architecture

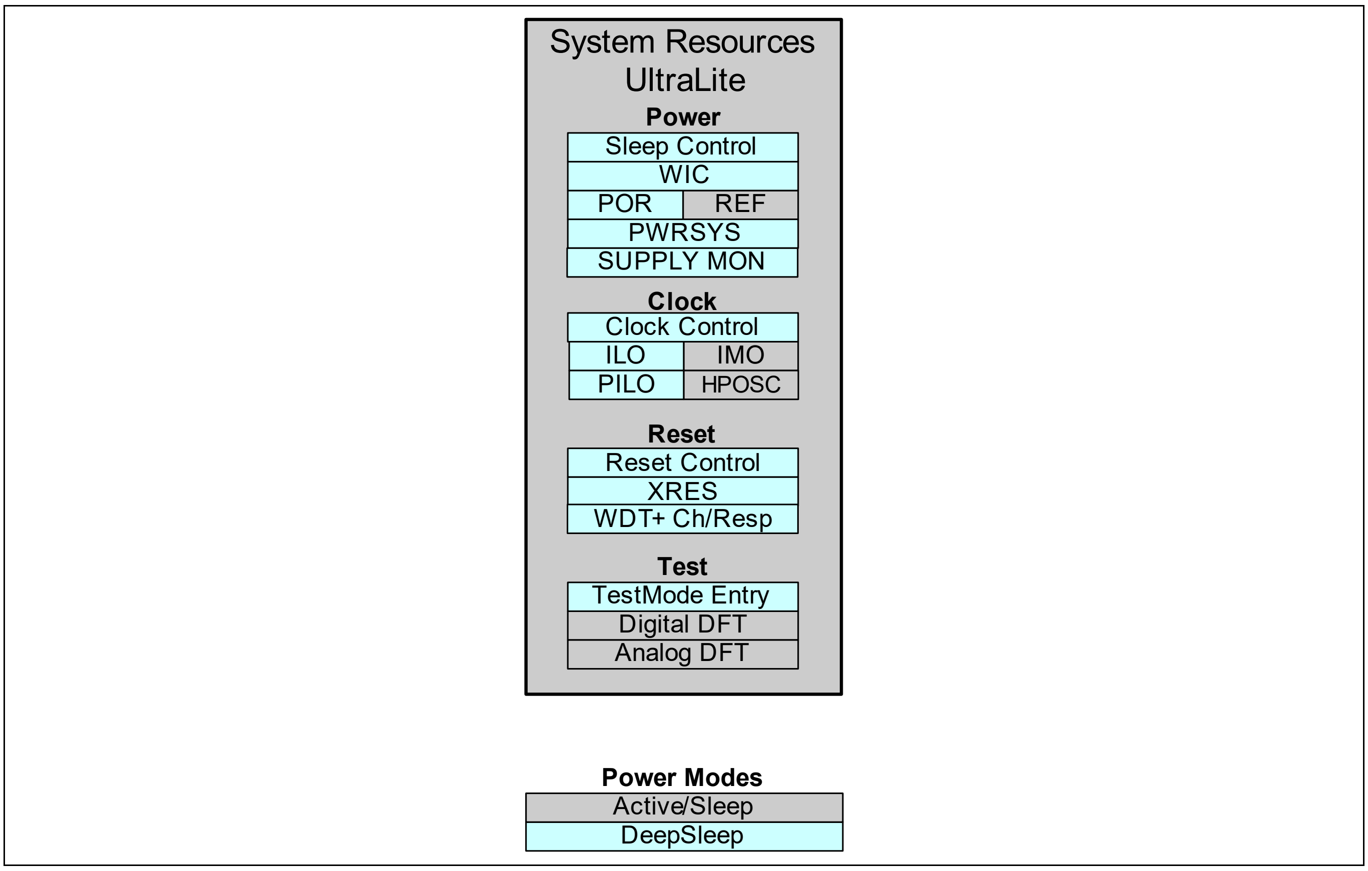


Figure 10-1. System-Wide Resources block diagram

11 Clocking system

The PSoC™ 4 HV PA clock system includes four internal clocks and one external clock:

- Four internal clock sources:
  - ±2% up to 49.152 MHz Internal Main Oscillator (IMO)
  - ±1% 2 MHz High-Precision Oscillator (HPOSC)
  - 40-kHz Internal Low-speed Oscillator (ILO)
  - ±5% or ±7% 32-kHz Precision Internal Low-speed Oscillator (PILO) based on the part number
- ±1% or ±1.5% accuracy on IMO and PILO when software calibrated to the HPOSC based on the part number
- One external clock source
  - External clock (EXTCLK) generated using a signal from an I/O pin
- PUMP clock (clk_pump) of 44 MHz to 49.152 MHz, sourced by IMO or HFCLK
- clk_hf (HFCLK) with divider by IMO, HPOSC or EXTCLK
- clk_lf (LFCLK) by ILO or PILO
- Dedicated prescaler for system clock (SYSCLK) allowing power vs. performance optimization
- Calibration counters to compare the frequency of two clock sources
- Four 16-bit and two 16.5 fractional dividers for accurate peripheral clocking
- Eight digital and analog peripheral clock destinations

11.1 Block diagram

Figure 11-1 gives a generic view of the clocking system in PSoC™ 4 HV PA devices.

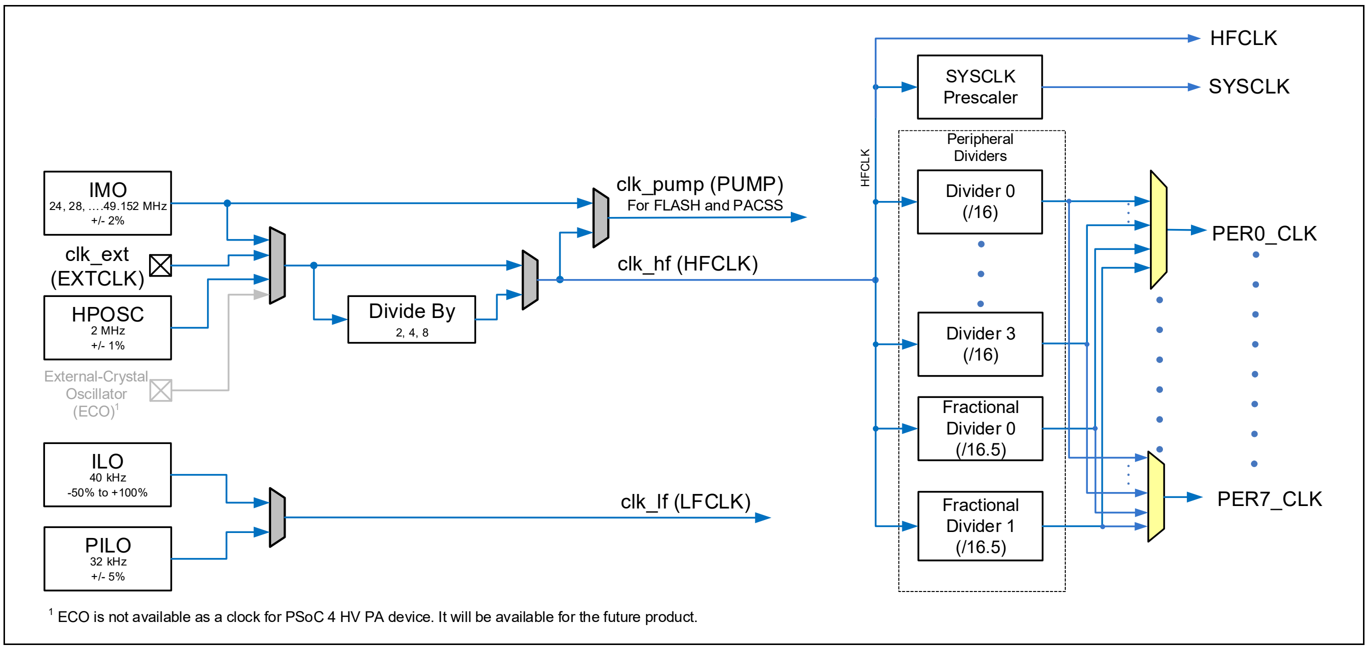


Figure 11-1. Clocking system block diagram

The five clock sources in the device are IMO, HPOSC, EXTCLK, ILO, and PILO, as shown in **Figure 11-1**. Note that ECO is not available as a clock for PSoC™ 4 HV PA device. PSoC™ 4 HV PA handles basic clock generation and safe clock switching for the main system high-speed clock (HFCLK). The HFCLK can be selected between the internal main oscillator (IMO), a high-precision oscillator (HPOSC), an external clock (EXTCLK). An included internal low-speed oscillator (ILO) and a precision internal low-speed oscillator (PILO) provides the low speed clock (LFCLK).

11.2 Clock sources

11.2.1 Internal main oscillator

The IMO is an accurate, high-speed internal (crystal-less) oscillator that is available as the main clock source during Active and Sleep modes. It is the default clock source for the device. Its frequency can be changed in 4-MHz steps between 24 MHz and 48 MHz. The 48-MHz setting can be boosted to 49.152 MHz using "special" calibration data stored in SFlash.

The IMO frequency is changed using the CLK_IMO_SELECT register. The default frequency is 24 MHz.

Table 11-1. IMO frequency

CLK_IMO_SELECT[2:0]	Nominal IMO Frequency
0	24 MHz
1	28 MHz
2	32 MHz
3	36 MHz
4	40 MHz
5	44 MHz
6	48 MHz/49.152 MHz

To get the accurate IMO frequency, trim registers are provided:

- OFFSET field in CLK_IMO_TRIM1 provides coarse trimming with a step size of 120 kHz.
- FSOFFSET field in CLK_IMO_TRIM2 is for fine trimming with a step size of 15 kHz.
- TCTRIM field in CLK_IMO_TRIM2 is for temperature compensation.
- STEPSIZE field in CLK_IMO_TRIM3 is to tune the step size of the FSOFFSET and OFFSET trims. These bits are determined at manufacturing time.

Trim settings are generated during manufacturing for every frequency that can be selected by CLK_IMO_SELECT. The 49.152 MHz is generated as "special" calibration data. These trim settings are stored in SFlash.

The trim settings are loaded during device startup; however, firmware can load new trim values and change the frequency in run time. Follow the algorithm in **Figure 11-2**. Note that the SFLASH trim data of 49.152 MHz is stored in SFLASH_IMO_xxx_LT7.

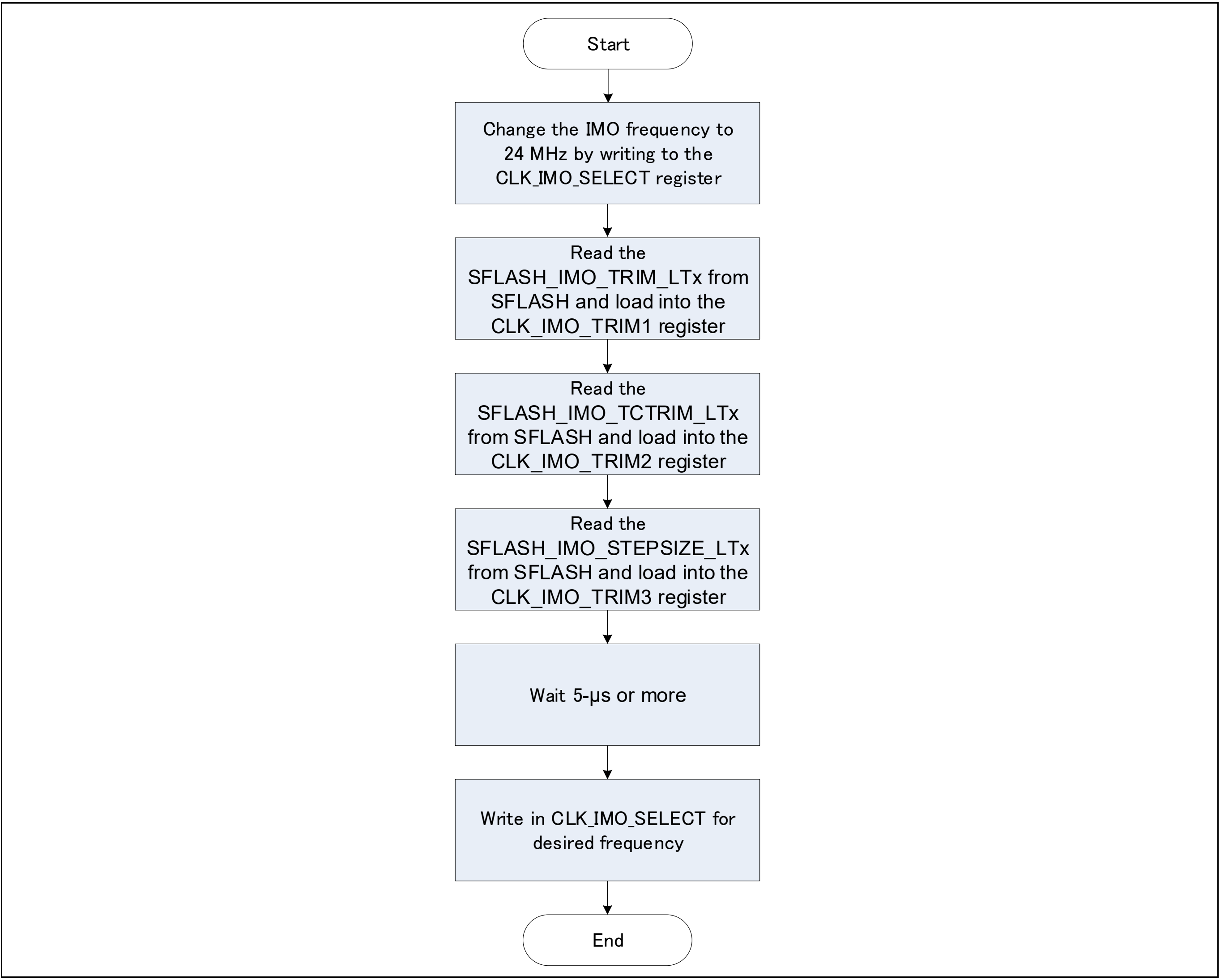

Start
Change the IMO frequency to 24 MHz by writing to the CLK_IMO_SELECT register
Read the SFLASH_IMO_TRIM_LTx from SFLASH and load into the CLK_IMO_TRIM1 register
Read the SFLASH_IMO_TCTRIM_LTx from SFLASH and load into the CLK_IMO_TRIM2 register
Read the SFLASH_IMO_STEPSIZE_LTx from SFLASH and load into the CLK_IMO_TRIM3 register
Wait 5-μs or more
Write in CLK_IMO_SELECT for desired frequency
End

Figure 11-2. Change IMO Frequency

To dynamically trim the IMO frequency, they should program both OFFSET field (120-kHz step size) in CLK_IMO_TRIM1 and FSOFFSET field (15-kHz step size) in CLK_IMO_TRIM2 via software.

Note that software must update the trim code properly when changing the clock frequency. Because there is no hardware safeguard that prevents the clock from exceeding the maximum frequency supported by the logic.

11.2.1.1 Startup behavior

After reset, the IMO is configured for 24-MHz with a prescaler of divide by 4 enabled. The system clock is 6 MHz after boot. During the “boot” portion of startup, trim values are read from flash and the IMO is configured to achieve datasheet specified accuracy.

11.2.1.2 Flash programming clock (PUMP)

PUMP clock is required for flash programming. This clock must be set to 48 MHz ± 4 MHz to program the flash. It is used to drive the charge pumps of the flash and for program/erase timing purposes. Pump clock allows setting the IMO at ~49.152 MHz while running the system at a slower frequency (divided by 2, 4, or 8).

A 48 MHz ± 4 MHz pump clock must be used, even for part numbers that only support 24 MHz system operation. The pump clock can be disabled when flash programming is not required.

11.2.1.3 High-precision oscillator

The high-precision internal oscillator (HPOSC) provides ±1% accuracy at 2 MHz. This oscillator is intended to trim other blocks to maintain 1% accurate timing throughout the chip. The calibration can be achieved using on-chip counters to compare clock rates and make appropriate adjustments via firmware.

The HPOSC operates from the low-voltage (1.8 V) V_{CCD} supply, and uses an available 1.2 µA current for fast startup. After startup, or when disabled, this current is not used and is cut off. The block is low power, typically using less than 10 µA.

The HPOSC is controlled with the HPOSC_CTL register.

11.2.1.4 Trim information

HPOSC uses trim inputs to achieve flat frequency response over temperature and a total accuracy of 0.1% of the target clock frequency. **Table 11-2** shows the HPOSC trim registers.

Important note: Trim registers are loaded by boot, so customer software should not set or change these registers.

Table 11-2. HPOSC Trim Registers

Registers	Bits	Mode
TRIM_HPOSC2_CTL	PTAT[7:0]	PTAT fine trim
TRIM_HPOSC0_CTL	PTATEF[1:0]	PTAT extra fine trim
TRIM_HPOSC0_CTL	PTATEC[1:0]	PTAT extra coarse trim
TRIM_HPOSC3_CTL	CTAT[7:0]	CTAT fine trim
TRIM_HPOSC0_CTL	CTATEF[1:0]	CTAT extra fine trim
TRIM_HPOSC0_CTL	CTATEC[1:0]	CTAT extra coarse trim
TRIM_HPOSC1_CTL	TCF[5:0]	Tempco - fine trim
TRIM_HPOSC4_CTL	TOC[2:0]	Tempco - coarse trim

11.2.2 Internal low-speed oscillator

The internal low-speed oscillator (ILO) operates with no external components and outputs a clock at 40-kHz nominal. The ILO is relatively low power, and low accuracy with ±40 percent, not suitable for precision timing or real-time clocking. The ILO is available in all power modes. The ILO is enabled and disabled with register CLK_ILO_CONFIG bit ENABLE.

11.2.3 Precision internal low-speed oscillator

The precision internal low-speed oscillator (PILO) provides ±5% or ±7% accuracy at 32 kHz (based on the part number). It is primarily used as the system low-frequency clock (LFCLK) when the ILO accuracy is insufficient. This block can be trimmed to better than 1% accuracy under any operating condition, using its 0.2% (or better) trim resolution. The user will need to adjust some trim (TR_CAP field in PILO_CTL) to meet the ±1% accuracy target. Typical current consumption is 650 nA from a V_{CCD} (1.8 V nominal) supply.

11.2.3.1 Trim information

PILO trim involves matching the tempco of the V_{REF} voltage to the capacitor charging currents to maintain flat frequency response over temperature. In addition, the output frequency's absolute value is set with coarse and fine trim controls. To meet the ±1% accuracy target, the user need to trim PILO_CTL.TR_CAP to lock to HPOSC using the **"Clock calibration counters"** on page 98. The MSB [15] of TR_CAP must always equal 1 when doing the calibration (TR_CAP will be > 128. The trim uses only the lower 7 bits). This register is initialized at reset with a default value, which sets the PILO to its nominal frequency and is updated with firmware for tracking. This allows adjusting the PILO output clock frequency, for example, with dynamic calibration against an accurate clock to compensate drift due to temperature or supply voltage change. The resolution is less than 0.2%, and increasing values give a monotonic increase in PILO frequency. The trim registers are summarized in **Table 11-3**.

Important note: Trim registers are loaded by boot, so customer software should not set or change these registers except the PILO_CTL.TR_CAP register.

Table 11-3. PILO Trim Registers

Registers	Bits	Mode
PILO_CTL	TR_CAP	PILO user temperature fine trim. The register is lock protected so the register must be unlocked before changing values.
TRIM_PILO2_CTL	PTAT[7:0]	PTAT fine trim
TRIM_PILO0_CTL	PTATEF[1:0]	PTAT extra fine trim
TRIM_PILO0_CTL	PTATEC[1:0]	PTAT extra coarse trim
TRIM_PILO3_CTL	CTAT[7:0]	CTAT fine trim
TRIM_PILO0_CTL	CTATEF[1:0]	CTAT extra fine trim
TRIM_PILO0_CTL	CTATEC[1:0]	CTAT extra coarse trim
TRIM_PILO1_CTL	TCF[5:0]	Tempco - fine trim
TRIM_PILO4_CTL	TOC[2:0]	Tempco - coarse trim

11.2.4 External clock

The external clock (EXTCLK) is a MHz range clock that can be generated from a signal on a designated PSoC™ 4 HV PA pin. This clock may be used instead of the IMO as the source of the system high-frequency clock, HFCLK. The allowable range of external clock frequencies is 1 MHz–48 MHz. The device always starts up using the IMO and the external clock must be enabled in user mode.

When manually configuring a pin as the input to the EXTCLK, the drive mode of the pin must be set to high-impedance digital to enable the digital input buffer. See the **"I/O system"** on page 141 for more details.

11.3 Clock calibration counters

A feature of the clocking system in PSoC™ 4 HV PA device is built-in hardware calibration counters. These counters can be used to compare the frequency of two clock sources. The primary use case is to take a higher accuracy clock such as the HPOSC and use it to measure a lower accuracy clock such as the IMO and PILO. The result of this measurement can then be used to trim both IMO and PILO (alternately). The calibration counters are available only in Active/Sleep modes. Counter values are not retained and will be in default state after wakeup from Deep Sleep.

There are two counters: Counter 1 is clocked off the Calibration Clock 1 and counts down; Counter 2 is clocked off the Calibration Clock 2 and counts up. When Counter 1 reaches 0, Counter 2 stops counting up and its value can be read. From that value the frequency of Calibration Clock 2 can be determined with the following equation:

$$\text{Calibration Clock 2} = \text{Calibration Clock 1} \times \frac{\text{Counter 2}}{\text{Counter 1}}$$

For example, if Calibration Clock 1 = 2 MHz, Counter 1 = 2500, and Counter 2 = 41

Calibration Clock 2 = 2 MHz × (41/2500) = 32.8 kHz.

Calibration Clock 1 and Calibration Clock 2 are selected with the CLK_DFT_SELECT register. All clock sources are available as a source for these two clocks.

Counter 1 is programmed in CLK_CAL_CNT1. Calibration counting starts immediately after a non-zero value is loaded. When Calibration Counter 1 reaches 0, CLK_CAL_CNT1.CAL_COUNTER_DONE is set. Then Counter 2 can be read in CLK_CAL_CNT2 (See **Table 11-4**).

The maximum error of the calibration measurement is estimated from the formula:

$$\text{Error} = \frac{(\text{Calibration Clock 1} / \text{Calibration Clock 2})}{\text{Counter 1}}$$

Error is reduced when a larger value of Counter 1 is used, subject to the limitation that both Counter 1 and Counter 2 are 16-bit counters without any overflow protection.

The calibration counters can be used to monitor clocks for drift or failure provided that the system clock is still functioning. It is also possible to use the counters to dynamically update trim values for certain clock sources to improve frequency accuracy. First or second order frequency locking can be done using the following formula:

Trim(N+1) = Trim(N) + Gain1 × FrequencyError(N+1) (first order)

Trim(N+1) = Trim(N) + Gain1 × FrequencyError(N+1) + Gain2 × ΣFrequencyError(N:2) (second order)

where:

- FrequencyError = (Ideal Counter 2 - Counter 2) × Calibration Clock 1 / Counter 1
- ΣFrequencyError(N:2) is the sum of previous frequency errors, ignoring one (or more) initial measurements

The first order frequency locking is simpler, and is preferred when calibration is performed occasionally or irregularly. The second order frequency locking will drive the long-term error towards zero, but it assumes that the clock frequency is constant between trim updates, and that the calibration measurements are regularly spaced.

The Gain1 and Gain2 factors depend on the trim sensitivity of the clock source. Recommended values are:

- IMO: Gain1 = 50 LSB/MHz, Gain2 = 5 LSB/MHz
- PILO: Gain1 = 15000 LSB/MHz, Gain2 = 1500 LSB/MHz

For example:

- Calibration Clock 1 = HPOSC (2 MHz)
- Counter 1 = 2000
- Calibration Clock 2 = IMO (48 MHz)
- Ideal Counter 2 = 49152
- Trim(0) = 80

  FrequencyError(1) = (49152 – 48000) × 2 / 2000 = 1.152 MHz

  Trim(1) = 80 + 50 × 1.152 + 5 × 0 = 137.6 -> 137

  FrequencyError(2) = (49152 – 49139) × 2 / 2000 = 0.013 MHz

  Trim(2) = 137 + 50 × 0.013 + 5 × 0 = 137.65 -> 137

  FrequencyError(3) = (49152 – 49139) × 2 / 2000 = 0.013 MHz

  Trim(3) = 137 + 50 × 0.013 + 5 × 0.013 = 137.72 -> 137

  …

  Trim[7) = 137 + 50 × 0.013 + 5 × 0.078 = 138.04 -> 138

In this example, the first order equation will stay at a trim setting of 137, while the second order equation will vary between 137 and 138. This example ignores the impacts of jitter and calibration error; actual behavior will be noisier.

Table 11-4. Clock calibration counter registers

Register	Bit Field and Bit Name	Description
CLK_CAL_CNT1	CAL_COUNTER_DONE	Status bit indicating that the internal counter #1 is finished counting and CLK_CAL_CNT2.COUNTER stopped counting up.
	CAL_COUNTER1	Down-counter clocked on clock output #0. This register always reads as zero. Counting starts internally when this register is written with a nonzero value. CAL_COUNTER_DONE goes immediately low to indicate that the counter has started and will be asserted when the counters are done. Do not write this field unless CAL_COUNTER_DONE==1. Both clocks must be running or the measurement will not complete. A stalled counter can be recovered by selecting valid clocks, waiting until the measurement completes, and discarding the first result.
CLK_CAL_CNT2	CAL_COUNTER2	Up-counter clocked on clock output #1. When CLK_CAL_CNT1.CAL_COUNTER_DONE==1, the counter is stopped and can be read by SW. Do not read this value unless CAL_COUNTER_DONE==1. The expected final value is related to the ratio of clock frequencies used for the two counters and the value loaded into counter 1: CLK_CAL_CNT2.COUNTER=(F_cnt2/F_cnt1)*(CLK_CAL_CNT1.COUNTER)

11.4 Clock distribution

PSoC™ 4 HV PA clocks are developed and distributed throughout the device, as shown in **Figure 11-1**. The distribution configuration options are as follows:

- HFCLK/PUMP input selection
- LFCLK input selection
- SYSCLK prescaler configuration
- Peripheral divider configuration

11.4.1 HFCLK/PUMP input selection

Figure 11-4 shows the selection options for HFCLK and PUMP.

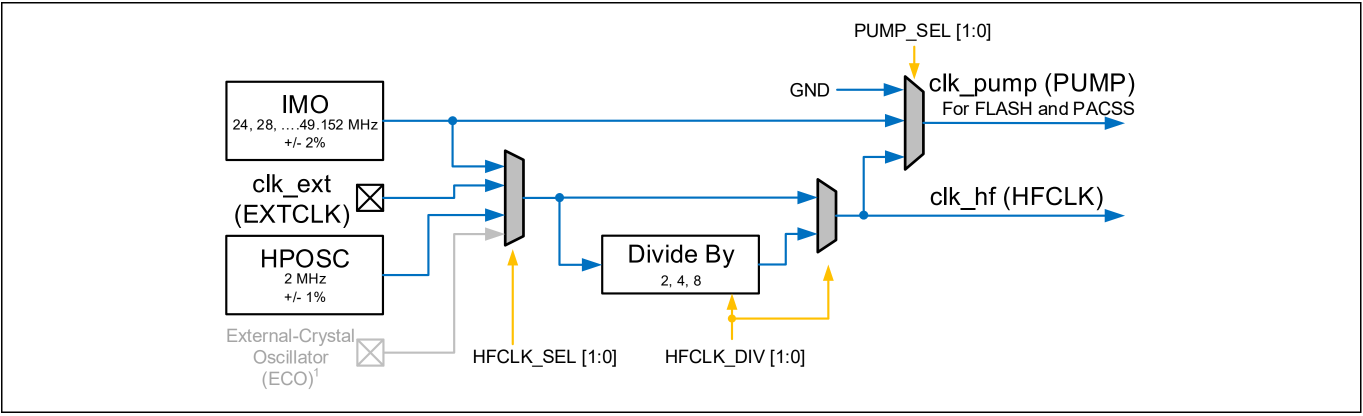


Figure 11-3. HFCLK selection options

11.4.1.1 HFCLK input selection

HFCLK in PSoC™ 4 HV PA has three input options: IMO, EXTCLK, and HPOSC. The HFCLK input is selected using the CLK_SELECT register's HFCLK_SEL bits, as described in **Table 11-5**.

Table 11-5. HFCLK input selection Bits HFCLK_SEL

Name	Description
HFCLK_SEL[1:0]	HFCLK input clock selection 0x0: IMO. Internal R/C Oscillator 0x1: EXTCLK. External Clock Pin 0x2: Reserved (ECO for future device) 0x3: HPOSC. High Precision Oscillator

Predivider is provided for HFCLK to limit the peak current of the device. The divider options are 2, 4, and 8 configured using CLK_SELECT register's HFCLK_DIV bits, as described in **Table 11-6**.

Table 11-6. HFCLK divider selection Bits HFCLK_DEV

Name	Description
HFCLK_DIV[3:2]	Selects clk_hf predivider value. It takes 3–4 clock cycles to switch to the new predivder value 0x0: NO_DIV: Transparent mode, feed through selected clock source w/o dividing. 0x1: DIV_BY_2: Divide the selected clock source by 2 0x2: DIV_BY_4: Divide the selected clock source by 4 0x3: DIV_BY_8: Divide the selected clock source by 8

Note:

- HFCLK_SEL and HFCLK_DIV settings should be separated

- There are 3–4 clock cycles to switch to the new predivider value after changing of HFCLK_DIV setting, wait for 4 cycles or more in the current HFCLK source frequency if needed.

11.4.1.2 PUMP input selection

clk_pump (PUMP) in PSoC™ 4 HV PA has two input options: IMO and HFCLK. The PUMP input is selected using the CLK_SELECT register's PUMP_SEL bits, as described in **Table 11-7**.

Table 11-7. PUMP input selection Bits PUMP_SEL

Name	Description
PUMP_SEL[1:0]	PUMP input clock selection 0: GND. No clock, connect to gnd 1: IMO. Use main IMO output 2: HFCLK. Use HFCLK (using selected source after predivider but before prescaler) 3: Reserved.

11.4.2 LFCLK input selection

Figure 11-4 shows the selection options for LFCLK.

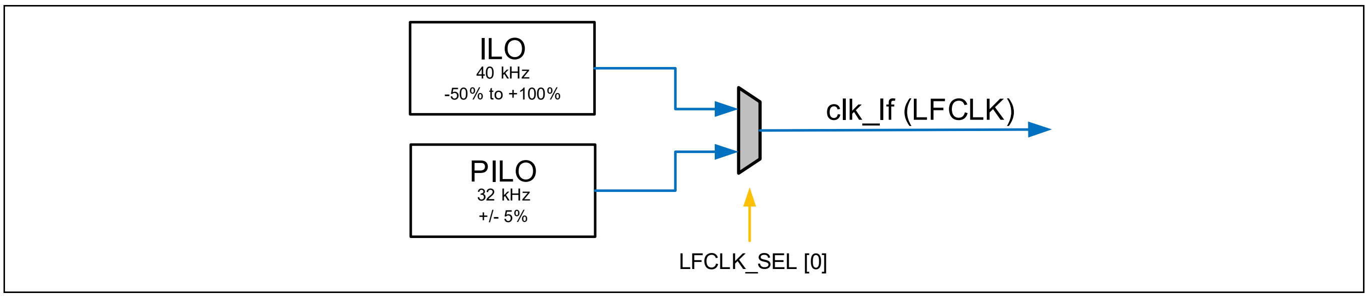


Figure 11-4. LFCLK selection options

LFCLK in PSoC™ 4 HV PA has two input options: ILO and PILO. The LFCLK input is selected using the CLK_SELECT register's LFCLK_SEL bits, as described in **Table 11-8**.

Table 11-8. LFCLK Input Selection Bits LFCLK_SEL

Name	Description
LFCLK_SEL[0]	LFCLK input clock selection 0: ILO. Internal Low Frequency Oscillator 1: PILO. Precision Internal Low Frequency Oscillator

11.4.3 SYSCLK prescaler configuration

The SYSCLK prescaler allows the device to divide the HFCLK before use as SYSCLK, which allows for non-integer relationships between peripheral clocks and the system clock. SYSCLK must be equal to or faster than all other clocks in the device that are derived from HFCLK. The SYSCLK prescaler is capable of dividing the HFCLK by powers of 2 between 2^0 = 1 and 2^3 = 8. The prescaler divide value is set using register CLK_SELECT bits SYSCLK_DIV, as described in **Table 11-9**. The prescaler is initially configured to divide by 1.

Table 11-9. SYSCLK prescaler divide value bits SYSCLK_DIV

Name	Description
SYSCLK_DIV[1:0]	SYSCLK prescaler divide value 0: SYSCLK = HFCLK 1: SYSCLK = HFCLK/2 2: SYSCLK = HFCLK/4 3: SYSCLK = HFCLK/8

11.4.4 Peripheral clock divider configuration

PSoC™ 4 HV PA has six clock dividers, which include four 16-bit clock dividers and two 16.5-bit fractional clock dividers. Fractional clock dividers allow the clock divisor to include a fraction of 0..31/32. The formula for the output frequency of a fractional divider is Fout = Fin / (INT16_DIV + 1 + (FRAC5_DIV/32)). For example, a 16.5-divider with an integer divide value of 3 (INT16_DIV=2, FRAC5_DIV = 0), produces signals to generate a 16-MHz clock from a 48-MHz HFCLK. A 16.5-divider with an integer divide value of 4 (INT16_DIV = 3, FRAC5_DIV = 0), produces signals to generate a 12-MHz clock from a 48-MHz HFCLK. A 16.5-divider with an integer divide value of 3(INT16_DIV = 2) and a fractional divider of 0.5 (FRAC5_DIV = 16) produces signals to generate a 13.7-MHz clock from a 48-MHz HFCLK. Not all 13.7-MHz clock periods are equal in size; half of them will be three HFCLK cycles and half of them will be two HFCLK cycles.

Fractional dividers are useful when a high-precision clock is required (for example, for a SPI serial interface). Fractional dividers are not used when a low jitter clock is required, because the clock periods have a jitter of 1 HFCLK cycle.

The divide value for each of the four integer clock dividers are configured with the PERI_DIV_16_CTLx registers and the two 16.5-bit fractional clock dividers are configured with the PERI_DIV_16_5_CTLx registers. **Table 11-10** and **Table 11-11** describe the configurations for these registers.

Table 11-10. Non-Fractional Peripheral Clock Divider Configuration Register PERI_DIV_16_CTLx

Bits	Name	Description
0	EN_x	Divider enabled. HW sets this field to ‘1’ as a result of an ENABLE command. HW sets this field to ‘0’ as a result on a DISABLE command.
23:8	INT16_DIV_x	Integer division by (1+INT16_DIV). Allows for integer divisions in the range [1, 65536].

Table 11-11. Fractional Peripheral Clock Divider Configuration Register PERI_DIV_16_5_CTLx

Bits	Name	Description
0	EN_x	Divider enabled. HW sets this field to ‘1’ as a result of an ENABLE command. HW sets this field to ‘0’ as a result on a DISABLE command.
7:3	FRAC5_DIV_x	Fractional division by (FRAC5_DIV/32). Allows for fractional divisions in the range [0, 31/32]. Note that fractional division results in clock jitter as some clock periods may be 1 “clk_hf” cycle longer than other clock periods.
23:8	INT16_DIV_x	Integer division by (1+INT16_DIV). Allows for integer divisions in the range [1, 65,536].

Each divider can be enabled using the PERI_DIV_CMD register. This register acts as the command register for all integer dividers and fractional dividers. The PERI_DIV_CMD register format is as follows.

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Description	Enable	Disable	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	PA_SEL_TYPE		PA_SEL_DIV						SEL_TYPE		SEL_DIV					

The SEL_TYPE field specifies the type of divider being configured. This field is ‘1’ for the 16-bit integer divider, ‘2’ for the 16.5-bit fractional divider.

The SEL_DIV field specifies the number of the specific divider being configured. For the integer dividers, this number ranges from 0 to 15. For fractional dividers, this field is any value in the range 0 to 3. When SEL_DIV = 63 and SEL_TYPE = 3, no divider is specified.

The (PA_SEL_TYPE, PA_SEL_DIV) field pair allows a divider to be phase-aligned with another divider. The PA_SEL_DIV specifies the divider which is phase aligned. Any enabled divider can be used as a reference. The PA_SEL_TYPE specifies the type of the divider being phase aligned. When PA_SEL_DIV = 63 and PA_SEL_TYPE = 3, HFCLK is used as a reference.

Consider a 48-MHz HFCLK and a need for a 12-MHz divided clock A and a 8-MHz divided clock B. Clock A uses a 16-bit integer divider 0 and is created by aligning it to HF_CLK ((PA_SEL_TYPE, PA_SEL_DIV) is (3, 63)) and DIV_16_CTL0.INT16_DIV is 3. Clock B uses the integer divider 1 and is created by aligning it to clock A ((PA_SEL_TYPE, PA_SEL_DIV) is (1, 0)) and DIV_16_CTL1.INT16_DIV is 5. This guarantees that clock B is phase-aligned with clock A as the smallest common multiple of the two clock periods is 12 HFCLK cycles, the clocks A and B will be aligned every 12 HFCLK cycles. Note that clock B is phase-aligned to clock A, but still uses HFCLK as a reference clock for its divider value.

Each peripheral block in PSoC™ has a unique peripheral clock (PERI#_CLK) associated with it. Each of the peripheral clocks have a multiplexed input, which can take the input clock from any of the existing clock dividers.

Table 11-12 shows the mapping of the mux output to the corresponding peripheral blocks (shown in **Figure 11-1**). Any of the peripheral clock dividers can be mapped to a specific peripheral by using their respective PERI_PCLK_CTLx register.

Table 11-12. PSoC™ 4 HV PA Devices Peripheral Clock Multiplexer Output Mapping

PERI#_CLK	Peripheral
0	SCB0
1	LIN0
2	LIN1
3	TCPWM0
4	TCPWM1
5	TCPWM2
6	TCPWM3
7	PACSS

Table 11-13. Programmable Clock Control Register - PERI_PCLK_CTLx

Bits	Name	Description
1:0	SEL_DIV	Specifies one of the dividers of the divider type specified by SEL_TYPE. If SEL_DIV is 63 and SEL_TYPE is 3 (default/reset value), no divider is specified and no clock control signal(s) are generated.
7:6	SEL_TYPE	Specifies divider type: 0: Reserved. 1: 16.0 (integer) clock dividers. 2: 16.5 (fractional) clock dividers. 3: Reserved.

11.5 Low-Power mode operation

The high-frequency clocks including the IMO, EXTCLK, HPOSC, HFCLK, SYSCLK and peripheral clocks operate only in Active and Sleep modes. The ILO, PILO and LFCLK operate in all power modes.

For lowest possible power consumption in Deep Sleep mode, the ILO and PILO can both be disabled. In this case, either a GPIO or another peripheral (that is, LIN PHY) must be used to wake-up the part. Refer to the **"Interrupts"** on page 54 for more details on wakeup sources.

11.6 Register List

Table 11-14. Clocking System Register List

Register name	Description
SRSSHV	
CLK_IMO_CONFIG	IMO Configuration Register - This register controls the IMO configuration (lock protected)
CLK_IMO_SELECT	IMO Frequency Select Register (lock protected)
CLK_ILO_CONFIG	ILO Configuration Register - This register controls the ILO configuration (lock protected)
HPOSC_CTL	High Precision Oscillator Control Register (lock protected)
PILO_CTL	Precision Low Power Oscillator Control Register (lock protected)
CLK_SELECT	Clock Select Register - This register controls clock tree configuration, selecting different sources for the system clocks (lock protected)
CLK_IMO_TRIM1	IMO Trim Register - This register contains IMO trim for frequency trim bits (lock protected)
CLK_IMO_TRIM2	IMO Trim Register - This register contains IMO trim for temperature compensation trim and frequency trim bits (lock protected)
CLK_IMO_TRIM3	IMO Trim Register - This register contains the IMO trim step-size bits (lock protected)
PWR_BG_TRIM1	Bandgap Trim Registers - These registers control the trim of the bandgap reference, allowing manipulation of the voltage references in the device (lock protected).
PWR_BG_TRIM2	
TRIM_HPOSCx_CTL	High Precision Oscillator Trim Control Registers (lock protected)
TRIM_PILOx_CTL	Low Frequency Oscillator Trim Control Registers (lock protected)
CLK_CAL_CNT1	Clock Calibration Counter 1 Register
CLK_CAL_CNT2	Clock Calibration Counter 2 Register
CLK_DFT_SELECT	Clock DFT Mode Selection Register
PELI	
PERI_DIV_CMD	Peripheral Clock Divider Command Registers
PERI_PCLK_CTLx	Programmable Clock Control Registers - These registers are used to select the input clocks to peripherals.
PERI_DIV_16_CTLx	Peripheral Clock Divider Control Registers - These registers configure the peripheral clock dividers, setting integer divide value, and enabling or disabling the divider.
PERI_DIV_16_5_CTLx	Peripheral Clock Fractional Divider Control Registers - These registers configure the peripheral clock dividers, setting fractional divide value, and enabling or disabling the divider.
SFLASH.128x4	
SFLASH_IMO_TRIM_LTx	IMO Frequency Trim Register
SFLASH_IMO_TCTRIM_LTx	IMO Temperature Compensation Trim
SFLASH_IMO_STEPSIZE_LTx	IMO Stepsize Trim

Table 11-14. Clocking System Register List (continued)

Register name	Description
SFLASH_IMO_4PCT_LIM	MO trim 4% deviation from nominal
SFLASH_IMO_3PCT_LIM	IMO trim 3% deviation from nominal
SFLASH_PILO_6PCT_LIM	PILO trim 6% deviation from nominal

12 Power supply and monitoring

The power system consists of regulators to generate appropriate voltages. The PSoC™ 4 HV PA operates at full performance from a single supply on V_{BAT} over a voltage range of 3.6 V to 28 V and remains functional up to 42 V. The system has the following regulators:

- High-Voltage (HV) Regulator
- Active Digital Regulator
- Deep-Sleep Regulator

12.1 Block diagram

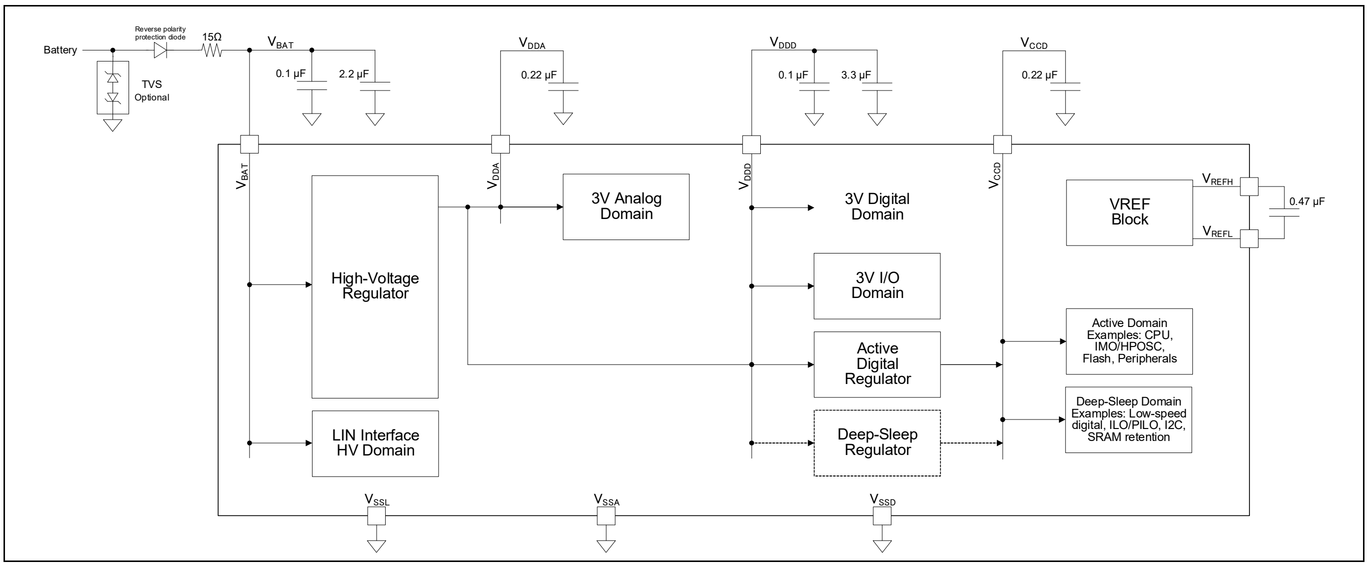


Figure 12-1. Power System block diagram

Figure 12-1 shows the power system diagram and all the power supply pins.

The high-voltage regulator generates a 3.3-V supply from V_{BAT} for V_{DDD} and V_{DDA}. V_{DDA} powers analog circuits, while V_{DDD} provides power for I/Os (GPIOs). The active digital regulator and deep-sleep regulator generate a 1.8-V supply from V_{DDD} for V_{CCD} to support the various power modes. V_{CCD} powers digital circuits.

12.2 Power supply

Figure 12-2 shows the PSoC™ 4 HV PA power supply input.

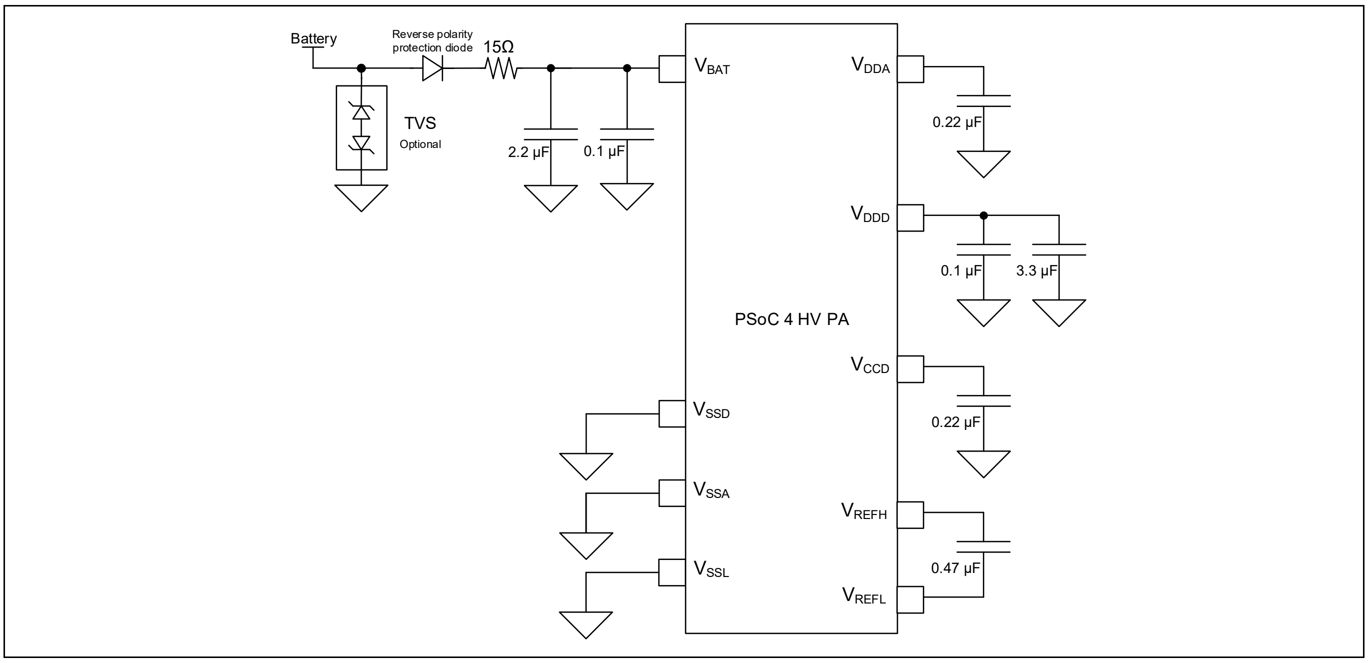


Figure 12-2. PSoC™ 4 HV PA Power Supply Input

Bypass capacitors must be used from V_{BAT}, V_{DDD}, V_{DDA}, and V_{CCD} to ground, and between V_{REFH} and V_{REFL}. These capacitors should typically be X7R ceramic or better. The V_{BAT} bypass capacitors must be 50-V rated voltage or more.

Table 12-1. Bypass capacitors

Power Supply	Bypass Capacitor
$V_{BAT} - V_{SSD}$	0.1-µF ceramic plus 2.2-µF bypass capacitor
$V_{DDD} - V_{SSD}$	0.1-µF ceramic plus 3.3-µF ceramic always required
$V_{DDA} - V_{SSA}$	0.22-µF ceramic bypass capacitor
$V_{CCD} - V_{SSD}$	0.22-µF ceramic bypass capacitor
$V_{REFH} - V_{REFL}$	0.47-µF ceramic bypass capacitor

Battery input filter resistor (typical 15 Ω) will add between a reverse polarity protection diode and the bypass capacitors. The resistor power rating should be 0.5 W or more.

Reverse polarity protection diode protects the reverse current from V_{BAT} bypass capacitors. The forward voltage should be as low as possible.

Transient Voltage Suppressor (TVS) protects surge voltage such as load dump waveform from battery. This component is optional.

12.3 How it works

The regulators in **Figure 12-1** power the various domains of the device. The system has a high-voltage (HV) regulator, which generates 3.3-V supplies and several regulators for various low-voltage core domains. The analog circuits run directly from the V_{DDA} supply generated by the HV regulator. The core regulators include an Active digital regulator for digital circuitry, and a separate regulator for Deep Sleep. The deep-sleep regulator has switches to pass high-power regulator voltages to loads when the low-power regulators are not required.

The HV regulator is always enabled. The Active digital regulator is enabled during the Active or Sleep power modes. It is turned off in the Deep Sleep power mode. The Deep Sleep regulator fulfills power requirements in the low-power modes.

Table 12-2. Details

Mode	HV Regulator	Active Regulator	Deep Sleep Regulator
Active	On	On	On
Sleep	On	On	On
Deep Sleep	On	Off	On

12.4 Voltage monitoring

The power supply includes supervision and monitoring to ensure that the required voltage levels exist for the respective modes. The voltage monitoring system includes power-on-reset (POR), brownout detection (BOD) and over-voltage detection (OVD). The supervisor either delays mode transitions (on POR, for example) until required voltage levels are achieved for proper function or generates resets (BOD, OVD) as appropriate.

12.4.1 Power-on-reset (POR)

POR circuits provide a reset pulse during the initial power ramp. POR circuits monitor V_{CC} (core voltage). The POR guarantees that all circuits are properly initialized before release. POR circuits are used during initial chip power-up and then disabled.

12.4.2 Brownout detection (BOD)

The BOD circuit protects the operating or retaining logic from possibly unsafe supply conditions by applying reset to the device. There are three BOD circuits that are the V_{DDD} detection (BODVDDD), V_{CCD} detection (BODVCCD) and High-Voltage Subsystem detection (BODHVSS) which is only available for DeepSleep supervision. Note that BODVDDD and BODVCCD are not available in DeepSleep mode.

The BOD circuit generates a reset if core voltage dips below the minimum safe operating voltage. The system will not come out of RESET until the supply is detected to be valid again.

To enable firmware to distinguish a normal power cycle from a brownout event, a special register is provided (RES_CAUSE), which will not be cleared after a BOD generated RESET. However, this register will be cleared if the device goes through POR or XRES.

12.4.3 Over-voltage detection (OVD)

PSoC™ 4 HV PA offers two OVD circuits that are V_{DDD} detection (OVDVDDD) and V_{CCD} detection (OVDVCCD).Similar to the BOD circuit, the OVD circuit detects supply conditions above a threshold and applies a reset. As the name suggests, the OVD circuit maintains a device reset, if V_{DDD} or V_{CCD} supply stays higher than thresholds. The OVD circuit can generate a reset in all device power modes except RESET_PORVDDD and RESET_XRES. For more information of power modes, refer to **"Power modes"** on page 113.

Note that the OVDVDDD and OVDVCCD are not available in DeepSleep mode.

12.4.4 Controls BOD and OVD

The PWR_SSV_CTL register controls the brownout and over-voltage detectors.

BODVDDD and BODVCCD settings are enable by default. Note that they cannot be disabled during normal mode.

OVDVDDD and OVDVCCD are disabled during boot-up. It can be enabled by writing to OVDVDDD/OVDVCCD_ENABLE fields.

BODHVSS is also disabled during boot-up. It can be enabled by writing to BODHVSS_ENABLE field. The enable allows the BOD to take effect, it does not disable the detector. The detector is in HVSS and is always on.

12.4.5 Monitoring supplies with ADC

The ADCs can measure supply voltages (V_{DDD}, V_{DDA}, V_{CCD}, V_{SSD}, and V_{SSA}) pins for diagnostic purposes. See **"Input multiplexer"** on page 322 for more information.

12.5 Voltage references

The SRSS includes a bandgap and current references for use by analog circuits, flash, and several system resources blocks. The high-voltage regulator is a standalone circuit with an internal voltage reference. The precision analog channel subsystem (PACSS) has a separate high-precision voltage reference, which provides accurate voltage references for the ADCs.

12.6 Register list

Table 12-3. Power supply and monitoring register list

Register name	Description
PWR_CONTROL	Power Mode Control Register (Lock Protected)
PWR_KEY_DELAY	Power System Key & Delay Register (Lock Protected)
PWR_SSV_CTL	Supply Supervisory Control Register (Lock Protected)
PWR_SSV_STATUS	Supply Supervision Status Register
RES_CAUSE	Reset Cause Observation Register
PWR_PWRSYS_TRIM1	Power System Trim Register (Lock Protected)
PWR_BG_TRIM1	Bandgap Trim Register (Lock Protected)
PWR_BG_TRIM2	Bandgap Trim Register (Lock Protected)
TRIM_BOD	Brown Out Detect Trim Register (Lock Protected)
TRIM_OVD	Over Voltage Detect Trim Register (Lock Protected)

13 Chip operational modes

PSoC™ 4 HV PA is capable of executing firmware in four different modes. These modes dictate execution from different locations in flash and ROM, with different levels of hardware privileges. Only three of these modes are used in end-applications; debug mode is used exclusively to debug designs during firmware development.

PSoC™ 4 HV PA operational modes are:

- Boot
- User
- Privileged
- Debug

13.1 Boot

Boot mode is an operational mode where the device is configured by instructions hard-coded in the device SROM. This mode is entered after the end of a reset, provided no debug-acquire sequence is received by the device. Boot mode is a privileged mode; interrupts are disabled in this mode so that the boot firmware can set up the device for operation without being interrupted. During boot mode, hardware trim settings are loaded from flash to guarantee proper operation during power-up. When boot concludes, the device enters user mode and code execution from flash begins. This code in flash may include automatically generated instructions from the IDE that will further configure the device.

At the end of Boot, the content of the CPUSS_SYSREQ register defines Boot status. Each bit in the CPUSS_SYSREQ register defines specific Boot status. The decoding of the bits is present in **Table 13-1** Boot Statuses.

A zero value for CPUSS_SYSREQ after boot implies a successful boot, while non-zero values should be compared with **Table 13-1** to determine the cause and severity. The CPUSS_BOOT_RESULT_0 and CPUSS_BOOT_RESULT_1 registers contain details of logged Boot error. If a few errors are logged in the CPUSS_SYSREQ register, then CPUSS_BOOT_RESULT_0/1 contains details of the logged error with the highest priority.

There is a group of Boot statuses with Special priority. Other logged Boot statuses are cleared once a status with special priority is set.

Table 13-1. Boot statuses

Priority	CPUSS_SYSREQ Bit Number	Status Description[a)]	CPUSS_BOOT_RESULT_0	CPUSS_BOOT_RESULT_1
High	0	Flash read error	X	X
	1	Calibration error. Trims are not applied correctly. Reboot is recommended.	Address which failed on write	X
	2	Invalid SFLASH checksum	Calculated checksum	Correct checksum from SFLASH
	3-4	Reserved	–	–
	5	FAULT source 0	FAULT_Data[0] register	FAULT_Data[1] register
	6	FAULT source 1		
	7–18	FAULT sources 2–13		
	19	FAULT source 14		
	20	FAULT source 15		
Low	21–26	Reserved	–	–

Table 13-1. Boot statuses (continued)

Priority	CPUSS_SYSREQ Bit Number	Status Description[a)]	CPUSS_BOOT_RESULT_0	CPUSS_BOOT_RESULT_1
Special	27	HardFault	Value of PC register before HardFault	Value of LR register before HardFault
	28-31	Reserved	-	-

a) Logged Faults could be detected by the last Boot process or by Boot/ Application before the warn reset. Boot detects only NC ECC Faults for (S)Flash and SRAM. The application can detect all Fault sources.

13.2 User

User mode is an operational mode where normal user firmware from flash is executed. User mode cannot execute code from SROM. Firmware execution in this mode includes the automatically generated firmware by the IDE and the firmware written by the user. The automatically generated firmware can govern both the firmware startup and portions of normal operation. The boot process transfers control to this mode after it has completed its tasks.

13.3 Privileged

Privileged mode is an operational mode, which allows execution of special subroutines that are stored in the device ROM. These subroutines cannot be modified by the user and are used to execute proprietary code that is not meant to be interrupted or observed. Debugging is not allowed in privileged mode.

The CPU can transition to privileged mode through the execution of a system call. For more information on how to perform a system call, see **"Performing a system call"** on page 376. Exit from this mode returns the device to user mode.

13.4 Debug

Debug mode is an operational mode that allows observation of the PSoC™ 4 HV PA operational parameters. This mode is used to debug the firmware during development. The debug mode is entered when an SWD debugger connects to the device during the acquire time window, which occurs during the device reset. Debug mode allows IDEs to debug the firmware. Debug mode is only available on devices in open mode (one of the four protection modes). For more details on the debug interface, see the **"Program and debug interface"** on page 361.

For more details on protection modes, see the **"Device security and register protection"** on page 137.

14 Power modes

The PSoC™ 4 HV PA provides three power modes, intended to minimize the average power consumption for a given application. The power modes, in the order of decreasing power consumption, are:

- Active
- Sleep
- Deep Sleep

Active, Sleep, and Deep Sleep are standard Arm®-defined power modes, supported by the Arm® CPUs

The power consumption in different power modes is controlled by using the following methods:

- Enabling/disabling peripherals
- Powering on/off internal regulators
- Powering on/off clock sources
- Powering on/off other portions of the PSoC™ 4 HV PA

Figure 14-1 illustrates the various power modes and the possible transitions between them.

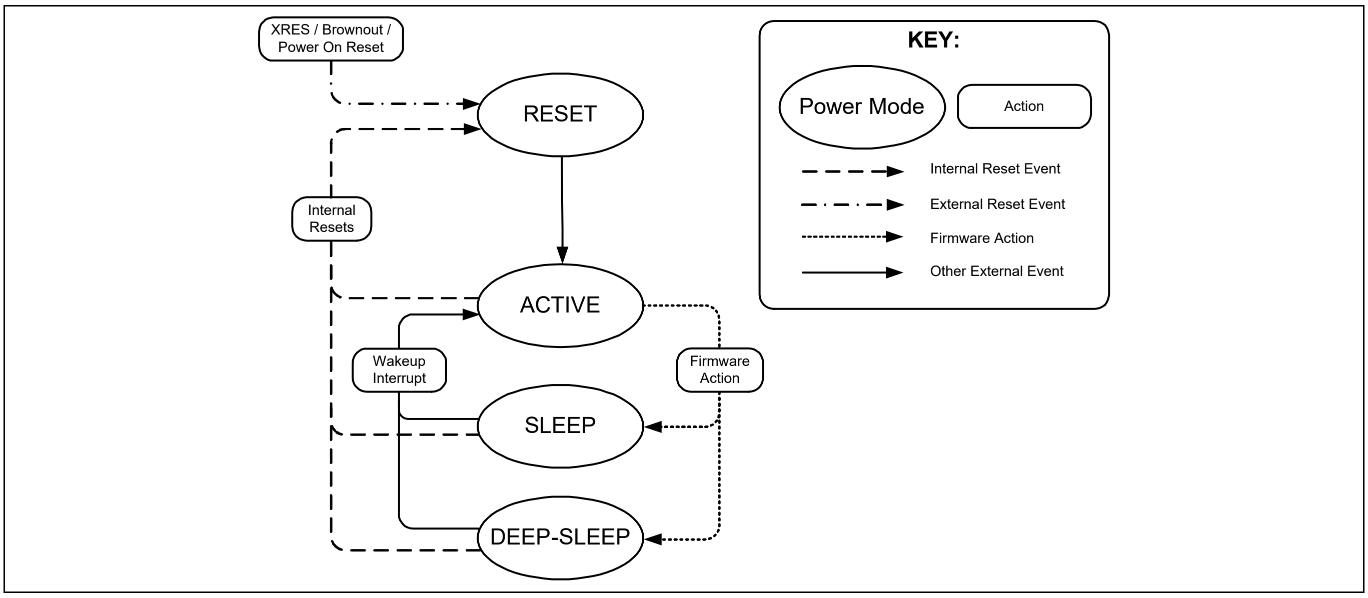


Figure 14-1. Power mode transitions state diagram

Note: Arm® nomenclature for Deep Sleep power mode is ‘SLEEPDEEP’.

Table 14-1 illustrates the power modes offered by PSoC™ 4 HV PA.

Table 14-1. PSoC™ 4 HV PA power modes

Power Mode	Description	Entry Condition	Wakeup Sources	Active Clocks	Wakeup Action	Available Regulators
Active	Primary mode of operation; all peripherals are available (programmable).	Wakeup from other power modes, internal and external resets, brownout, power on reset	Not applicable	All (programmable)	N/A	All regulators are available.
Sleep	CPU enters Sleep mode and SRAM is in retention; all peripherals are available (programmable).	Manual register write	Any enabled interrupt	All (programmable) except CPU clock	Interrupt	All regulators are available.
Deep Sleep	All internal supplies are driven from the deep sleep regulator. IMO and high-speed peripherals are off. Only the low-frequency clock is available. Interrupts from low-speed, asynchronous, or low-power analog peripherals can cause a wakeup.	Manual register write	GPIO interrupt, SCB, watchdog timer, lifetime counter, LIN PHY	PILO (32 kHz) ILO (40 kHz)	Interrupt	Deep-sleep regulator

14.1 Active mode

Active mode is the primary power mode of the PSoC™ device. This mode provides the option to use every possible subsystem/peripheral in the device. In this mode, the CPU is running and all the peripherals are powered. The firmware may be configured to disable specific peripherals that are not in use, to reduce power consumption.

14.2 Sleep mode

This is a CPU-centric power mode. In this mode, the Cortex®-M0+ CPU enters Sleep mode and its clock is disabled. It is a mode that the device should come to very often or as soon as the CPU is idle, to accomplish low power consumption. It is identical to Active mode from a peripheral point of view. Any enabled interrupt can cause wakeup from Sleep mode.

14.3 Deep Sleep mode

In Deep Sleep mode, the CPU, SRAM, and high-speed logic are in retention. The high-frequency clocks are disabled. Optionally, the PILO or ILO remains on and low-frequency peripherals continue to operate. Digital peripherals that do not need a clock or receive a clock from their external interface (for example, I^2C slave) continue to operate. Interrupts from low-speed, asynchronous, or low-power analog peripherals can cause a wakeup from Deep Sleep mode.

The available wakeup sources are listed in **Table 14-3**.

14.4 Power mode summary

Table 14-2 illustrates the peripherals available in each low-power mode; **Table 14-3** illustrates the wakeup sources available in each power mode.

Table 14-2. Available Peripherals

Peripheral	Active	Sleep	Deep Sleep[a)]
CPU	Available	Retention[b)]	Retention
SRAM	Available	Retention	Retention
DMA	Available	Available	Not Available
High-speed peripherals	Available	Available	Retention
Low-speed peripherals	Available	Available	Available (optional)
Internal main oscillator (IMO)	Available	Available	Not Available
High-Precision Oscillator (HPOSC)	Available	Available	Not Available
Precision Internal Low-speed Oscillator (PILO)	Available	Available	Available (optional)
Internal low-speed oscillator (ILO, 40 kHz)	Available	Available	Available (optional)
Asynchronous peripherals (peripherals that do not run on internal clock)	Available	Available	Available
Power-on-reset, Brownout detection	Available	Available	Available
GPIO output state	Available	Available	Available

a) All registers marked with the “NonRetention” flag return to their default states during Deep Sleep mode.

b) The configuration and state of the peripheral is retained. Peripheral continues its operation when the device enters Active mode.

Table 14-3. Wakeup Sources

Power Mode	Wakeup Source	Wakeup Action
Sleep	Any enabled interrupt source	Interrupt
	Any reset source	Reset
Deep Sleep	GPIO interrupt	Interrupt
	I^2C address match	Interrupt
	LIN wakeup interrupt	Interrupt
	Watchdog timer	Interrupt/Reset
	Lifetime counter	Interrupt

Note: In addition to the wakeup sources mentioned in **Table 14-3**, external reset (XRES) and power monitor reset bring the device to Active mode from any power mode. XRES and brownout trigger a full system restart. All the states including frozen GPIOs are lost. In this case, the cause of the reset will be stored in the SRSS reset cause register.

14.5 Deep Sleep Wakeup hold-off

The PWR_KEY_DELAY.WAKEUP_HOLDOFF register controls the delay in waiting for references to settle on wake up from DEEPSLEEP. BOD is ignored and the system does not resume until this delay expires. Note that the same delay on POR is hard-coded. The default assumes the output of the predivider is 48 MHz + 3%. Firmware may scale this setting according to the fastest actual clock frequency that can occur when waking from DEEPSLEEP.

The default value of PWR_KEY_DELAY.WAKEUP_HOLDOFF assumes a 48-MHz clk_hf (including the predivider, but not the prescaler). For different clk_hf settings, firmware can optimize the wake time by scaling the hold-off to achieve the same time.

It is functionally acceptable to leave the default hold-off setting, but DEEPSLEEP wakeup time may exceed the specification. The minimum allowed hold-off is ceiling (default setting × F_clk_hf/48).

A larger setting can be used to artificially increase the wake time. This is useful in giving additional settling time, for example, for failure analysis or functional purposes.

14.6 Low-Power mode entry and exit

A Wait For Interrupt (WFI) instruction from the Cortex®-M0+ (CM0+) triggers the transitions into Sleep and Deep Sleep mode. The Cortex®-M0+ can delay the transition into a low-power mode until the lowest priority ISR is exited (if the SLEEPONEXIT bit in the CM0 System Control register is set).

The transition to Sleep and Deep Sleep modes are controlled by the flags SLEEPDEEP in the CM0P System Control register (CM0P_SCR)

- Sleep is entered when the WFI instruction is executed, SLEEPDEEP = 0.
- Deep Sleep is entered when the WFI instruction is executed, SLEEPDEEP = 1.

Note: Since the CPU write buffers are not retained in DeepSleep power mode, ensure that they are empty prior to executing the WFI instruction for Deep sleep mode transition.

The LPM READY bit in the PWR_CONTROL register shows the status of the deep-sleep regulator. If the firmware tries to enter Deep Sleep mode before the regulators are ready, then PSoC™ 4 HV PA goes to Sleep mode first, and when the regulators are ready, the device enters Deep Sleep mode. This operation is automatically done in hardware.

In Sleep and Deep Sleep modes, a selection of peripherals are available (see **Table 14-3**), and firmware can either enable or disable their associated interrupts. Enabled interrupts can cause wakeup from low-power mode to Active mode. Additionally, any RESET returns the system to Active mode. See the **“Interrupts”** on page 54 and the **“Reset system and interrupts”** on page 133 for details.

14.7 Register list

Table 14-4. Power mode register list

Register name	Description
CM0P_SCR	System Control Register - Sets or returns system control data.
PWR_CONTROL	Power Mode Control Register (Lock Protected)
PWR_KEY_DELAY	Power System Key and Delay Register (Lock Protected)

15 Watchdog timer

The Watchdog Timers (WDTs) are used to automatically reset the device in the event of an unexpected firmware execution path. They are also used as a wakeup source to periodically generate interrupts as a wakeup source in low-power modes.

There are several Watchdog Timer functionalities in PSoC™ 4 HV PA devices. These are:

- Basic WDT
  - The Basic WDT is a free-running up-counter with programmable limit values and a maximum of 32-bit resolution
- Challenge-Response WDT (CRWDT)
  - The CRWDT includes a window watchdog function, generating timeout events if the CRWDT is serviced too soon, too late, or with the wrong software key. A register identifies the timeout cause. It generates a watchdog reset or interrupt if serviced too soon or too late. Service too soon potentially means an infinite loop including watchdog service is executing, while too late means the processor may be stuck and not processing properly. The challenge/response means the watchdog service routines must present specific data or "keys" in the order expected by the watchdog or a fault will occur. The fault will generate a watchdog reset or interrupt with the reset recorded in the Reset Cause register. The causes can be conditions such as watchdog too soon, watchdog late, or wrong key received.
- Lifetime Counter with wakeup capability.
  - The 32-bit lifetime counter includes a prescaler (/1 to /32) and triggered from the LFCLK clock. This counter runs in all modes and can be reset by POR. With the prescaler, the net resolution of the counter becomes 37-bit causing an overflow every 49.7 days. The counter will continue counting upon overflow.

15.1 Features

The WDTs have these features:

- System reset generation after a configurable interval
- Periodic interrupt/wake up generation in Active, Sleep, and Deep Sleep power modes
- Basic WDT
  - 32-bit counter and compares, clocked by LFCLK
  - Lower, Warn, and Upper limits with programmable actions for each
  - Implemented in V_{CCD} logic domain
- Challenge/response WDT (CRWDT)
  - Integrated Linear Feedback Shift Register (LFSR) updates when accessed
  - 24 bit counter and compares, clocked by LFCLK
  - Lower, Warn, and Upper limits, generates interrupts and faults
  - Implemented in V_{CCD} logic domain
- Lifetime counter
  - 32-bit free running counter, clocked by LFCLK with 1–32 prescalar
  - Up to every 49.7 days of roll over (Used to calculate total time the battery is connected)
  - Interrupt generation (Used to schedule mode transitions or other occasional actions)

15.2 Block diagram

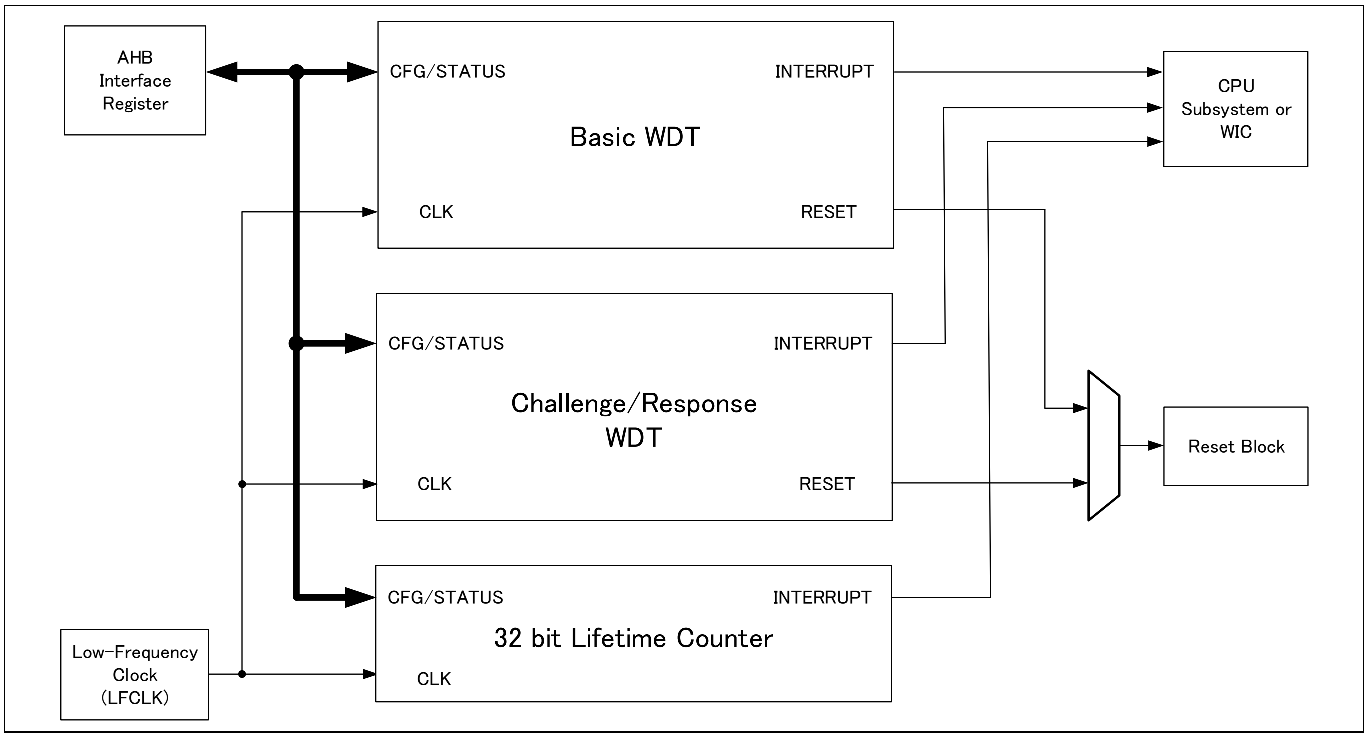


Figure 15-1. Watchdog timer block diagram

15.3 Basic WDT

15.3.1 Overview

The basic WDT is a free-running up-counter with programmable limit values, a maximum of 32-bit resolution, and a clock from the LFCLK. Servicing the watchdog clears and restarts the counter at zero.

The WDT can be configured to act on different counter limits where a reset is triggered if the watchdog is not serviced before the upper limit. In the window mode, a reset is triggered if the servicing occurs before the lower limit is reached. The warning limit triggers an interrupt to request servicing. Each of these actions can be activated independently. The WDT is enabled and specific registers are locked by default. An unlocking sequence is required to prevent accidental accesses. The WDT operates in Active, Sleep, and Deep Sleep modes. After a WDT reset the device returns to Active mode.

Figure 15-2 shows the functional overview of the WDT.

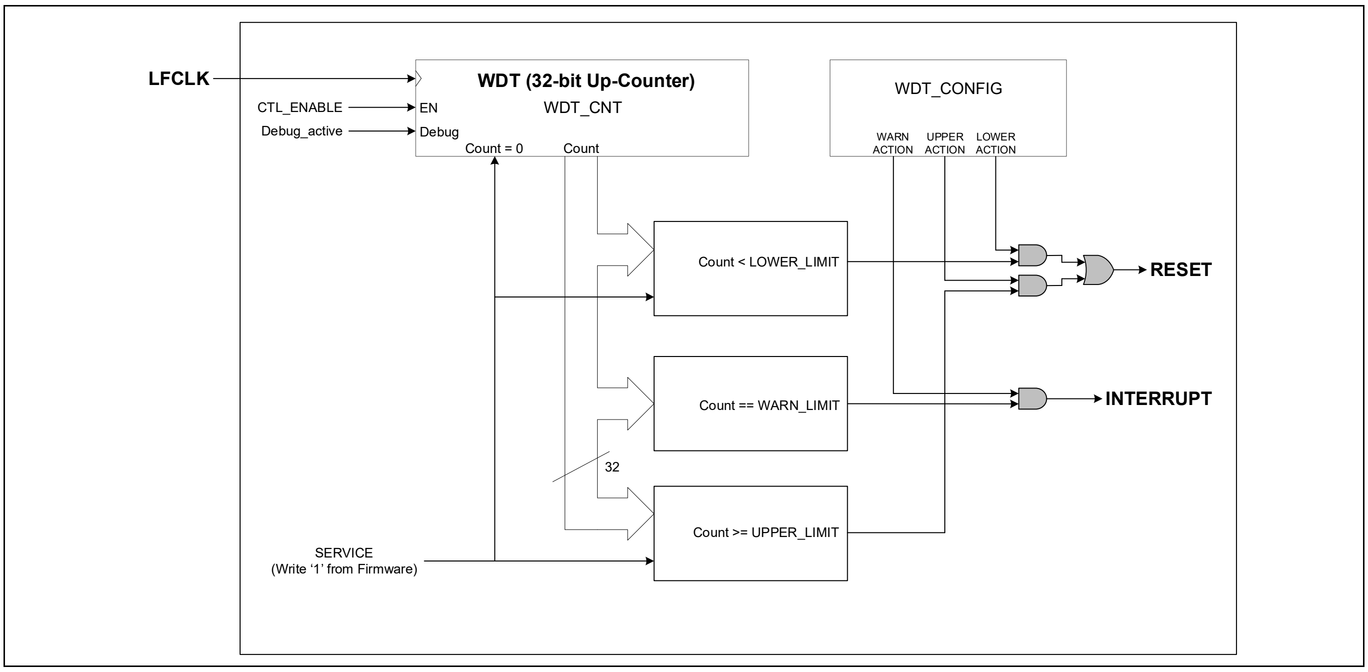


Figure 15-2. Basic WDT functional diagram

When enabled, the WDT counts up on each rising edge of the LFCLK clock. When the counter value (WDT_CNT register) equals the warning threshold value stored in WDT_WARN_LIMIT [31:0], an interrupt is generated if the WARN_ACTION [8] bit is set to '1' in the WDT_CONFIG register. The warn event will not reset the WDT counter and the WDT continues counting until it reaches the timeout threshold value stored in UPPER_LIMIT [31:0]; it generates a reset if the UPPER_ACTION [4] bit is set to '1' in the WDT_CONFIG register. If no action is taken on the upper threshold, the counter increments to the 32-bit boundary and then wraps around to '0' and counts up. In the window mode, an early threshold stored in LOWER_LIMIT [31:0] can be used if the LOWER_ACTION [0] bit is set to '1' in the WDT_CONFIG register for generating a reset if the counter is serviced too early. The watchdog counter is serviced by the SERVICE [0] bit in the WDT_SERVICE register. If this bit is set to '1' the watchdog counter is set to zero.

The WDT [0] bit in the WDT_INTR register is set whenever the WDT counter matches with the WARN_LIMIT and an interrupt is requested by the CPU. This interrupt must be cleared by writing a '1' to the same bit (WDT bit of WDT_INTR). Clearing the interrupt does not reset the watchdog counter.

The WDT can be enabled or disabled using the ENABLE [31] bit of the WDT_CTL register. The actual status of the counter is indicated by the ENABLED [0] bit of the WDT_CTL register.

The WDT provides a mechanism to lock WDT configuration registers. The WDT_LOCK bits [1:0] control the lock status of WDT-related registers. These are special bits, which can enable the lock in a single write (WDT_LOCK = 3); to release the lock, two different write accesses are required (WDT_LOCK = 1 to clear WDT_LOCK [0] and WDT_LOCK = 2 to clear WDT_LOCK [1]). When the WDT_LOCK bits are not equal to '0' the write accesses to the CTL, LOWER_LIMIT, WARN_LIMIT, UPPER_LIMIT, CNT, and SERVICE registers are prohibited. Note that this field is two bits to force multiple writes only. It represents only a single write protect signal protecting all those registers at the same time. WDT will lock and enable on any reset. This field is not retained during Deep Sleep mode, so the WDT will be locked after wakeup from these modes.

Note: The lock mechanism is an additional safety opportunity, which requires to unlock/lock the SERVICE register when servicing each watchdog counter.

When the watchdog counter is disabled and unlocked, the count value can be written for verification and debugging purposes. Software writes are always ignored when the counter is enabled.

Table 15-1 explains various registers and bit fields used to configure and use the WDT.

Table 15-1. Basic watchdog timer configuration options

Register [Bit_Pos]	Bit name	Description
WDT_CTL[31]	ENABLE	Enable or disable the watchdog counter • 0: Counter is disabled (not clocked) • 1: Counter is enabled (counting up)
WDT_CTL[0]	ENABLED	Indicates actual state of watchdog
WDT_LOCK[1:0]	WDT_LOCK	Prohibits writing control and configuration registers related to this MCWDT when not equal to 0 • 0: No effect • 1: Clear bit 0 • 2: Clear bit 1 • 3: Set both bit 0 and 1 (lock enabled)
WDT_CNT[31:0]	CNT	Current value of WDT counter
WDT_LOWER_LIMIT[31:0]	LOWER_LIMIT	Lower limit for watchdog
WDT_UPPER_LIMIT[31:0]	UPPER_LIMIT	Upper limit for watchdog
WDT_WARN_LIMIT[31:0]	WARN_LIMIT	Warn limit for watchdog

Table 15-1. Basic watchdog timer configuration options (continued)

Register [Bit_Pos]	Bit name	Description
WDT_CONFIG[0]	LOWER_ACTION	Action taken if this watchdog is serviced before LOWER_LIMIT is reached • 0: Do nothing • 1: Trigger a reset
WDT_CONFIG[4]	UPPER_ACTION	Action taken if this watchdog is not serviced before UPPER_LIMIT is reached • 0: Do nothing • 1: Trigger a reset
WDT_CONFIG[8]	WARN_ACTION	Action taken when the count value reaches WARN_LIMIT • 0: Do nothing • 1: Trigger an interrupt
WDT_CONFIG[12]	AUTO_SERVICE	Automatically service when the count value reaches WARN_LIMIT. This allows creation of a periodic interrupt if this counter is not needed as a watchdog.
WDT_CONFIG[28]	DEBUG_TRIGGER_EN	Enables the trigger input for the WDT to pause the counter in debug mode. • 0: Pauses the counter when a debug probe is connected. • 1: Pauses the counter when a debug probe is connected and the trigger input is high.
WDT_CONFIG[29]	DPSLP_PAUSE	Pauses/runs this counter when the system is in Deep Sleep • 0: Counter behaves normally during Deep Sleep • 1: Counter pauses during Deep Sleep
WDT_CONFIG[31]	DEBUG_RUN	Pauses/runs this counter while a debugger is connected • 0: Counter pauses according to DEBUG_TRIGGER_EN configuration • 1: Counter runs normally when debugger connected
WDT_INTR[0]	WDT	WDT Interrupt Request. This bit is set as configured by WDT action and limits. The WDT interrupt is cleared by writing a ‘1’ to this bit.
WDT_INTR_SET[0]	WDT	WDT Interrupt set register. Can be used to set interrupts for firmware testing.
WDT_INTR_MASK[0]	WDT	Mask for the WDT interrupt • 0: WDT interrupt is masked to CPU • 1: WDT interrupt is not masked to CPU
WDT_INTR_MASKED[0]	WDT	Logical AND of corresponding request and mask bits

15.3.2 Watchdog reset

A watchdog is typically used to protect the device against firmware/system crashes or faults. When the WDT is used to protect against system crashes, the WDT counter should be cleared by writing a '1' to the SERVICE [0] bit in the WDT_SERVICE register from a portion of the code that is not directly associated with the WDT interrupt. Otherwise, even if the main function of the firmware crashes or is in an endless loop, the WDT interrupt vector can still be intact and feed the WDT periodically.

The safest way to use the WDT against system crashes is to:

- Configure the UPPER_LIMIT such that firmware is able to reset the watchdog at least once during the period, even along the longest firmware delay path.
- In window mode, configure the LOWER_LIMIT to serve the watchdog counter not too early, even along the shortest firmware delay path.
- Reset (feed) the watchdog for clearing the counter regularly in the main body of the firmware code by setting the SERVICE [0] bit to '1' in WDT_SERVICE register.

It is not recommended to reset the watchdog counter in the WDT interrupt service routine (ISR), if WDT is being used as a reset source to protect the system against crashes. If necessary, use the warning interrupt to set a flag in the ISR. Local processing loops can observe that flag and break out of their loop. This allows the main loop to reach the servicing code (and clear the flag for the next pass through the main loop).

Recommended steps to use WDT as a reset source are as follows:

1. Write UPPER_LIMIT value to define the timeout period for reset generation. Set UPPER_ACTION [4] bit to '1' in the WDT_CONFIG register to enable a reset trigger when the watchdog counter reaches the UPPER_LIMIT.
2. If required, write the WARN_LIMIT to generate an interrupt before reaching the UPPER_LIMIT threshold. Do not use the ISR to feed the WDT; instead, use this interrupt to indicate that there is a firmware delay path, which is already critical. Use a warn level that is close enough to the UPPER_LIMIT but consider also the delay to handle the ISR and return to your main body functions for serving the watchdog counter. Set WARN_ACTION [8] bit to '1' in the WDT_CONFIG register to enable a watchdog warn interrupt when the watchdog counter matches with the WARN_LIMIT.
3. In window mode, define an adequate LOWER_LIMIT, which cannot be violated by the shortest firmware delay path. Set the LOWER_ACTION [0] bit to '1' in the WDT_CONFIG register to enable a reset trigger when the watchdog counter is serviced before the counter reaches the LOWER_LIMIT.
4. Set the WDT [0] bit in the WDT_INTR register to clear any pending WDT interrupt.
5. Enable low-speed oscillator either ILO (CLK_ILO_CONFIG.ENABLE) or PILO (PILO_CTL.ILO_EN).
6. Set the LFCLK_SEL in the CLK_SELECT register to select which clock source uses for LFCLK.
7. Enable the WDT by setting the ENABLE [31] bit in the WDT_CTL register.
8. In the firmware, write '1' to the SERVICE [0] bit in the WDT_SERVICE register to feed (reset) the watchdog.
9. Lock the WDT configuration by writing '3' to the WDT_LOCK bits.

Figure 15-3 shows all scenarios of the WDT operation while LOWER_ACTION, WARN_ACTION, and UPPER_ACTION are enabled.

- Counter is serviced between LOWER_LIMIT and WARN_LIMIT: This is the regular behavior of the WDT. No WARN interrupt is issued and no RESET is done.
- Counter is serviced between WARN_LIMIT and UPPER_LIMIT: The service is done late, a WARN interrupt is issued but no RESET is done.
- Counter is not serviced at all: WARN interrupt is issued but the SERVICE bit is not set. When the counter reaches the UPPER_LIMIT a reset is executed.
- Counter is serviced before the LOWER_LIMIT is reached: The counter is serviced too early; a reset is executed because the counter is cleared outside of the window.

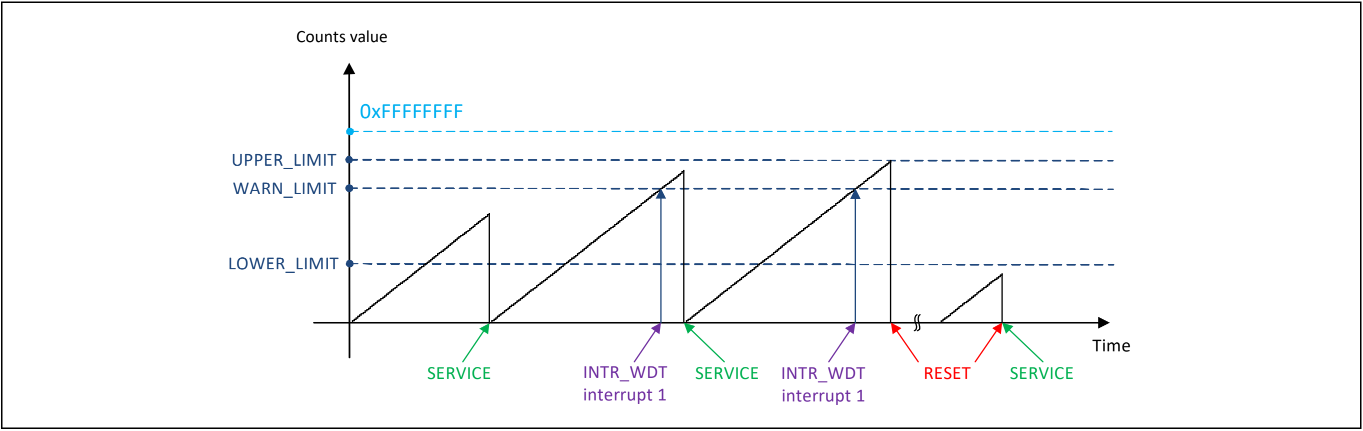


Figure 15-3. WDT counter operation in Window mode

Note: This figure illustrates the different scenarios with or without servicing the watchdog counter. It does not consider the WDT configuration, especially after a reset.

15.3.3 Watchdog interrupt

In addition to generating a device reset, the WDT can be used to generate interrupts. The watchdog counter can send interrupt requests to the CPU in Active power modes and to the wakeup interrupt controller (WIC) in Sleep and Deep Sleep power modes. It works as follows:

- Active Mode: In this mode, the WDT can send the interrupt to the CPU. The CPU acknowledges the interrupt request and executes the ISR. The interrupt must be cleared after entering the ISR in firmware.
- Sleep or Deep Sleep Mode: In these modes, the CPU subsystem is powered down. Therefore, the interrupt request from the WDT is directly sent to the WIC, which will then wake up the CPU. The CPU acknowledges the interrupt request and executes the ISR. The interrupt must be cleared after entering the ISR in firmware.

For more details on device power modes, see the **"Power modes"** on page 113. Follow these steps to use the WDT as a periodic interrupt generator:

1. Write the WARN_LIMIT to set the interrupt period. If the WDT is not serviced, the counter will continue to count up until the maximum counter level of 0xFFFFFFFF is reached and then the counter starts from zero.
2. Set the WARN_ACTION [8] bit to '1' in the WDT_CONFIG register to enable a watchdog warn interrupt when the watchdog counter matches with the WARN_LIMIT.
3. Set the WDT [0] bit in the WDT_INTR register to clear any pending WDT interrupt.
4. Enable the WDT interrupt to CPU by setting the WDT [0] bit in the WDT_INTR_MASK register.
5. Enable SRSS interrupt to the CPU by configuring the appropriate ISER register, see the **"Interrupts"** on page 54 for details.
6. In the ISR, clear the WDT interrupt; if required, clear the watchdog timer by writing '1' to the SERVICE [0] bit in the SERVICE register. Servicing the WDT allows to generate various interrupt periods, which can be defined by the WARN_LIMIT. Alternatively, set the AUTO_SERVICE[12] bit to '1' in the CONFIG register to automatically service the WDT when the count value reaches WARN_LIMIT.

Waking up from Deep Sleep mode requires to execute an unlock sequence by writing the value '1' to the WDT_LOCK [1:0] bits in the WDT_LOCK register followed by writing '2' to the same bit field.

Figure 15-4 shows the behavior of the WDT counter in interrupt mode. LOWER and UPPER actions are disabled. An interrupt is issued each time the counter matches the WARN_LIMIT and continuous to count up to the 32-bit maximum value. The interrupt period is calculated by 2^{32} x LFCLK clock cycles. The WDT does not provide an automatic counter clear function; therefore, the counter must be cleared manually by writing '1' to the SERVICE[0] bit in the WDT_SERVICE register.

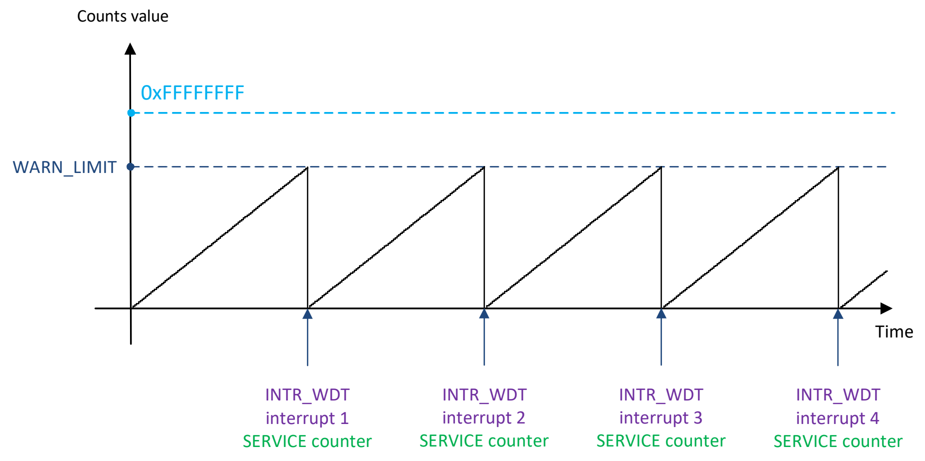


Figure 15-4. WDT Counter Operation with WARN Interrupt only

15.4 CRWDT

15.4.1 Overview

The CRWDT uses a challenge/response during a measured window to ensure that software is in control. The watchdog counters are clocked by LFCLK. The challenge and response values are provided by the Linear Feedback Shift Register (LFSR: CRC-8 AutoSAR) and is used to validate the time logged by the CRWDT_UPCNT. Firmware reads the LFSR value from CRWDT_CHALLENGE. The actions of reading the CRWDT_CHALLENGE or writing CRWDT_RESPONSE both increment the LFSR. The CRWDT is serviced by writing, in a timely manner, the value of CRWDT_RESPONSE that matches the current LFSR value. Successful servicing resets CRWDT_UPCNT to zero, starting a new counting period. The CRWDT also provides early and warning notifications.

Figure 15-5 shows the CRWDT block diagram.

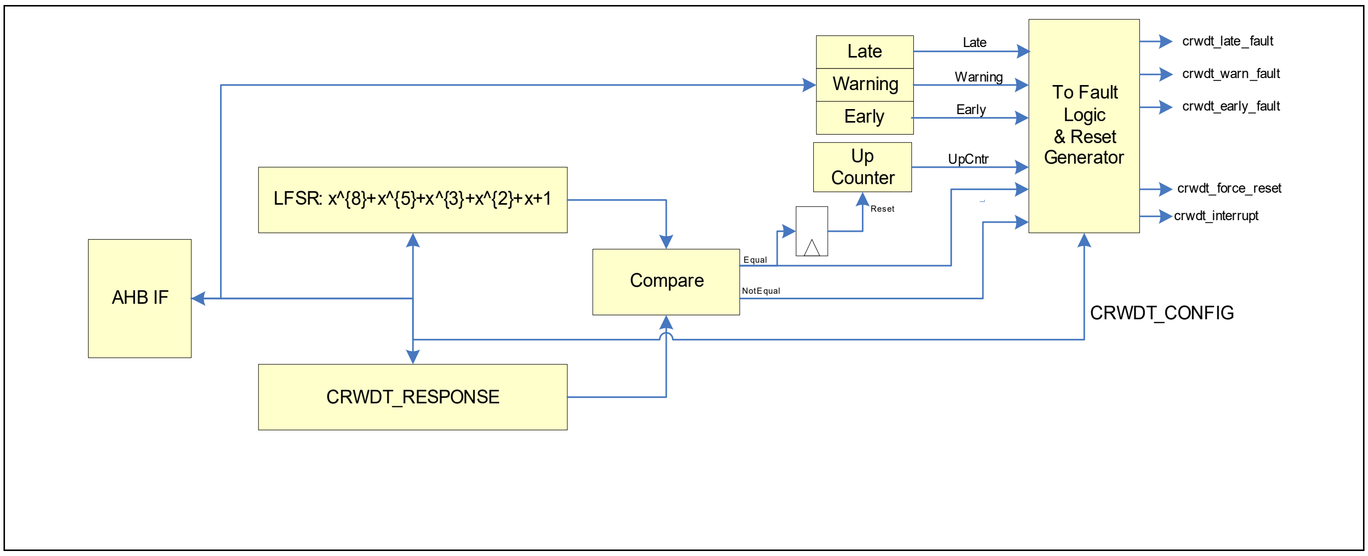


Figure 15-5. CRWDT block diagram

Table 15-2 explains various registers and bit fields used to configure and use the CRWDT.

Table 15-2. CRWDT Configuration options

Register [Bit_Pos]	Bit name	Description
CRWDT_CTL[31]	ENABLED	When set to "1", enables the Challenge/Response WatchDog Timer to Count. Requires 2 clk_lf cycles to take effect.
CRWDT_CTL[30]	STATUS_ENABLED	Indicates the actual state of CRWDT enable. May lag ENABLED by up to one clk_lf cycle.
CRWDT_CHALLENGE[7:0]	CHALLENGE	Challenge/Response WatchDog Challenge value. Implements the LFSR CCRC8-AUTOSAR using the polynomial x^8+x^5+x^3+x^2+x+1. The next value in the LFSR sequence is used to compare against the value subsequently written to the CRWDT_RESPONSE register. Incremented by write to CRWDT_RESPONSE to LFSR value next in the LFSR sequence after the value written to CRWDT_RESPONSE. Also incremented each time this register before a response is written to CRWDT_RESPONSE.

Table 15-2. **CRWDT Configuration options** (continued)

Register [Bit_Pos]	Bit name	Description
CRWDT_RESPONSE[7:0]	RESPONSE	Challenge/Response WatchDog Response value. Value is compared against the expected next value in the LFSR sequence following the value obtained from CRWDT_CHALLENGE. If the values match, the CRWDT_UPCNT resets. If the values miscompare, the action selected by CRWDT_CONFIG.CHALLENGE_FAIL_ACTION occurs.

CRWDT_UPCNT[23:0]	UPCNT	Challenge/Response WatchDog Up Counter. The counter up counts upon every clk_lf occurrence when enabled. If a match occurs between CRWDT_CHALLENGE and CRWDT_RESPONSE, the counter resets. If there is a mismatch, the action is taken selected by CRWDT_CONFIG.CHALLENGE_FAIL_ACTION. The counter will also reset when CRWDT_UPCNT=CRWDT_LATE.
		CRWDT_CTL.ENABLED = 0 - Counter reset
		CRWDT_CTL.ENABLED = 1 - Counter Increments

CRWDT_EARLY[23:0]	EARLY	Challenge/Response Early Value will cause the action selected by CRWDT_CONFIG.EARLY_ACTION if CRWDT_UPCNT < CRWDT_EARLY and there is a CRWDT_CHALLENGE/CRWDT_RESPONSE match. Writes to this register are ignored when CRWDT_CTL.ENABLED<>0.
CRWDT_WARN[23:0]	WARNING	Challenge/Response Warning Value will cause the action selected by CRWDT_CONFIGWARN_ACTION if CRWDT_UPCNT>CRWDT_WARN and there is a CRWDT_CHALLENGE/CRWDT_RESPONSE match. Writes to this register are ignored when CRWDT_CTL.ENABLED<>0.
CRWDT_LATE[23:0]	LATE	Challenge/Response Late Value will cause action selected by CRWDT_CONFIG.LATE_ACTION if CRWDT_UPCNT = CRWDT_LATE. Writes to this register are ignored when CRWDT_CTL.ENABLED<>0.

Table 15-2. CRWDT Configuration options (continued)

Register [Bit_Pos]	Bit name	Description
CRWDT_CONFIG[31]	DEBUG_RUN	Pauses/runs this counter while a debugger is connected. Other behaviors are unchanged during debugging, including service, configuration updates, and enable/disable. Note it may take up to two clk_lf cycles for the counter to pause and another two cycles to unpause, due to internal synchronization. If the debugger is connected for at least two clk_lf cycles, the EARLY_ACTION is ignored until after the first service after the debugger is disconnected. This prevents an unintentional trigger of the EARLY_ACTION before the firmware realigns the servicing period. After the first service, EARLY_ACTION behaves as configured. If the debugger is disconnected before two clk_lf cycles, the EARLY_ACTION may or may not be ignored. 0: When the debugger is connected, the counter pauses incrementing. 1: When the debugger is connected, the counter increments normally, but reset generation is blocked. Writes to this register are ignored when CRWDT_CTL.ENABLED<>0.
CRWDT_CONFIG[11]	CHALLENGE_FAIL_ACTION	Action is taken when a failed response occurs, i.e., the expected LFSR value is different than the expected value. Writes to this register are ignored when CRWDT_CTL.ENABLED<>0. 0x0 : NOTHING: Do nothing 0x1 : RESET: Trigger a reset
CRWDT_CONFIG[8]	WARN_ACTION	Action is taken if this watchdog when the proper response is written to CRWDT_RESPONSE and CRWDT_UPCNT reaches CRWDT_WARN and CRWDT_UPCNTF or CRWDT_WARN == CRWDT_UPCNT && CRWDT_LATE > CRWDT_UPCNT: The action is triggered on the same edge as when it meets this condition. For CRWDT_WARN Writes to this register are ignored when CRWDT_CTL.ENABLED<>0. 0x0: NOTHING: Do nothing 0x1: FAULT_AND_INT: Trigger a Fault and interrupt

Table 15-2. CRWDT Configuration options (continued)

Register [Bit_Pos]	Bit name	Description
CRWDT_CONFIG[4]	LATE_ACTION	Action is taken if this watchdog is not serviced before CRWDT_LATE is reached. The counter resets CRWDT_UPCNT when CRWDT_LATE is reached, regardless of CRWDT_CONFIG.LATE_ACTION setting. LATE_ACTION is ignored (i.e., treated as NOTHING) when a debugger is connected. For CRWDT_LATE == CRWDT_UPCNT: The action is triggered on the same edge as when it meets this condition. For CRWDT_LATE > CRWDT_UPCNT: No action is taken Writes to this register are ignored when CRWDT_CTL.ENABLED<>0. 0x0: NOTHING: Do nothing 0x1: FAULT_RESET: Trigger a fault. Further, trigger a system-wide reset if the CRWDT is not disabled within 6 clk_lf cycles.
CRWDT_CONFIG[0]	EARLY_ACTION	Action is taken if this watchdog when the proper response is written to CRWDT_RESPONSE before CRWDT_UPCNT reaches CRWDT_EARLY. EARLY_ACTION is ignored (i.e., treated as NOTHING) when a debugger is connected. For CRWDT_EARLY > CRWDT_UPCNT: The action is triggered on the same edge as when it meets this condition. For CRWDT_EARLY <= CRWDT_UPCNT: No action is triggered. Writes to this register are ignored when CRWDT_CTL.ENABLED<>0. 0x0: NOTHING: Do nothing 0x1: FAULT_AND_INT: Trigger a Fault and interrupt

15.4.2 Firmware usage

Figure 15-6 shows the CRWDT firmware usage.

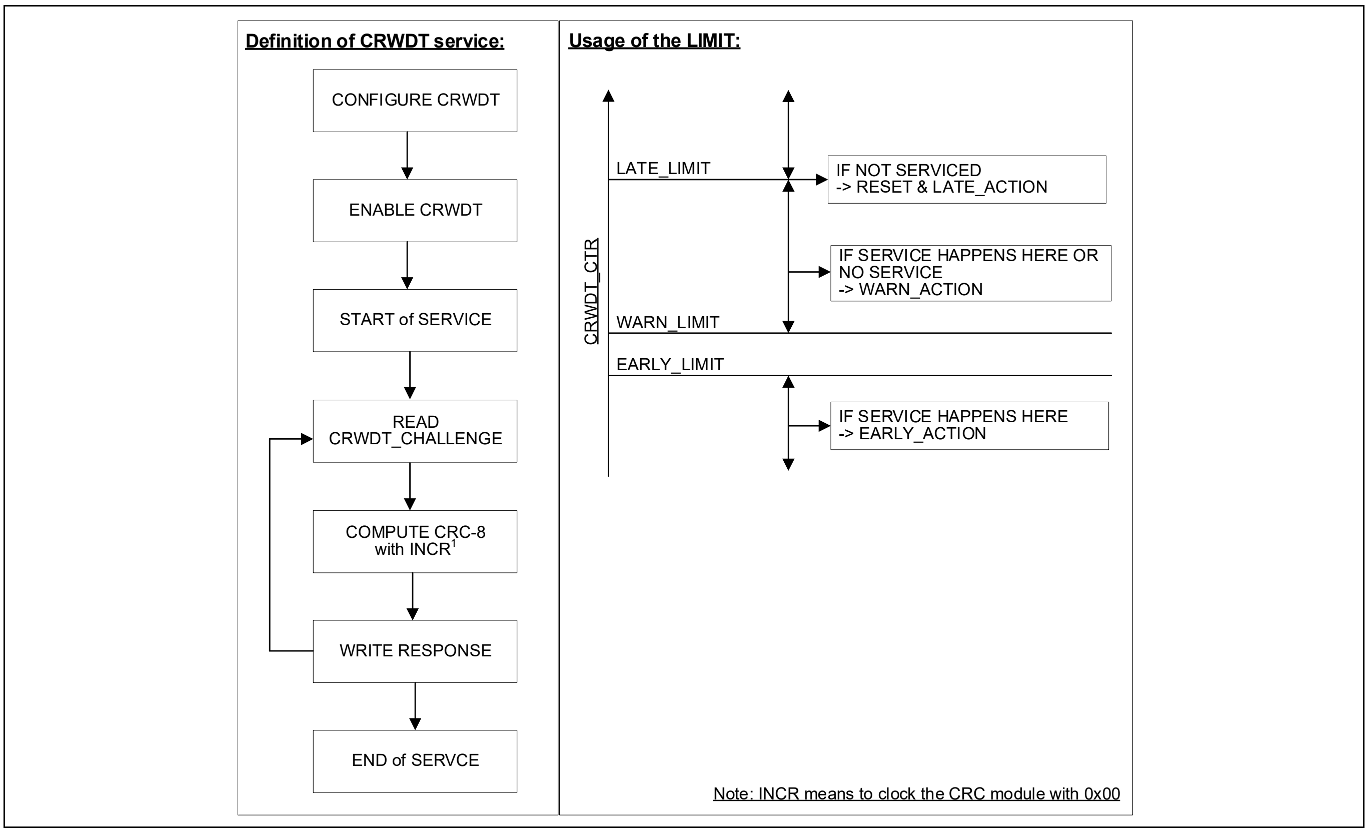

Definition of CRWDT service:
CONFIGURE CRWDT
ENABLE CRWDT
START of SERVICE
READ CRWDT_CHALLENGE
COMPUTE CRC-8 with INCR[1]
WRITE RESPONSE
END of SERVCE
Usage of the LIMIT:
CRWDT_CTR
LATE_LIMIT
IF NOT SERVICED -> RESET & LATE_ACTION
IF SERVICE HAPPENS HERE OR NO SERVICE -> WARN_ACTION
WARN_LIMIT
EARLY_LIMIT
IF SERVICE HAPPENS HERE -> EARLY_ACTION
Note: INCR means to clock the CRC module with 0x00

Figure 15-6. CRWDT Firmware usage

The steps recommended when using CRWDT are as follows:

1. Configure CRWDT

Write the DEBUG_RUN, CHALLENGE_FAIL_ACTION, WARN_ACTION, LATE_ACTION, EARLY_ACTION in the CRWDT_CONFIG register to set each action. For more information, see Table 15-2.

2. Enable CRWDT

Set the ENABLED bit to '1' in the CRWDT_CTL register to enable the CRWDT.

3. Start of service
4. Read CRWDT_CHALLENGE

Read the LFSR value from CHALLENGE bits in the CRWDT_CHALLENGE register. The action of reading the CHALLENGE increments the LFSR value.

5. Compute CRC-8 with INCR

The computed value will be written to the RESPONSE register at the appropriate time to service the CRWDT.

6. Write response
   a) If the response does not arrive before CRWDT_LATE = CRWDT_UPCNT, then LATE_ACTION occurs.
   b) If the response does arrive, but does not match the current LFSR value, then RESPONSE, CHALLENGE_FAIL_ACTION occurs.
   c) If the response does arrive and does match the current LFSR, then CRWDT_UPCNT is compared to the values of the CRWDT_WARN and CRWDT_EARLY registers.
      i. CRWDT_EARLY > CRWDT_UPCNT:
      EARLY_ACTION occurs
      ii. CRWDT_WARN <= CRWDT_UPCNT:
      WARN_ACTION occurs
7. Repeat 4 to 6
8. End of Service
   a) Set the ENABLED bit to ‘0’ in the CRWDT_CTL register to disable the CRWDT.

15.5 Lifetime counter

15.5.1 Overview

The lifetime counter provides a basic 32-bit counter operating in the Deep Sleep domain at a frequency lower than clk_lf (LFCLK) which can provide a interrupt when required. The lifetime counter includes a prescaler and triggered from the LFCLK clock. The prescalar provides the LFCLK from divide-by-2 up to divide-by-32.

The 32-bit counter runs off of the prescaled clock. It is capable of wrapping. The initial value may be loaded through the MMIO, and is intended to provide a life-cycle count that is maintained by software. It is expected that the value is loaded after POR by software, and software periodically updates Flash to the current count.

Figure 15-7 shows the Lifetime Counter Block Diagram.

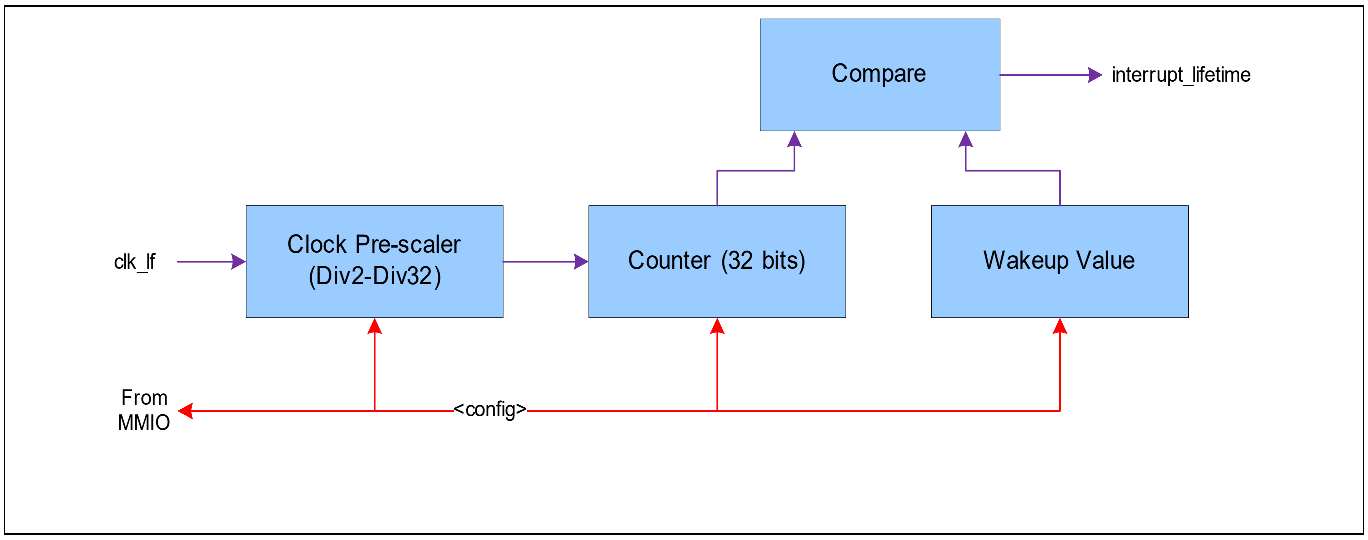


Figure 15-7. Lifetime counter block diagram

Table 15-3 explains various registers and bit fields used to configure and use the lifetime counter.

Table 15-3. Lifetime counter configuration options

Register [Bit_Pos]	Bit name	Description
LIFETIME_CTL[31]	ENABLED	When set to '1' enables LIFETIME_COUNTER to increment. Due to internal synchronization, it takes up to two LFCLK cycles to update the counters after a write to this register.
LIFETIME_CTL[30]	STATUS_ENABLED	Indicates actual state of lifetime counter enable. May lag ENABLED by up to one clk_lf cycles.
LIFETIME_CTL[2:0]	PRESEL	Select Divide ratio for preselector. Legal values are 0-4. 0x0: PRESEL_DIV2: Divide CLK_LF by 2 0x1: PRESEL_DIV4: Divide CLK_LF by 4 0x2: PRESEL_DIV8: Divide CLK_LF by 8 0x3: PRESEL_DIV16: Divide CLK_LF by 16 0x4: PRESEL_DIV32: Divide CLK_LF by 32
LIFETIME_WAKEUP[31:0]	WAKEUP	Compare the WAKEUP value against LIFETIME_COUNTER. If they are equal interrupt_wakeup to set. Due to internal synchronization, it takes up to two LFCLK cycles to update the counters after a write to this register.
LIFETIME_COUNTER[31:0]	COUNT	Lifetime counter which clocks on the output of the CLK_LF prescaler output controlled by LIFETIME_CTL.PRESEL. The counter does not increment unless LIFETIME_CTL.ENABLED =1. FW is responsible for initializing this value after reset and maintaining the running value.

15.5.2 Wakeup mechanism

The wakeup value is set in the MMIO and is compared against the 32-bit counter value. If there is a match, the interrupt_lifetime signal is asserted. This is a Deep Sleep interrupt.

15.6 Register list

Table 15-4. WDT registers

Register name	Description
Basic WDT (WDT_B_STRUCT)	
WDT_CTL	WDT Control Register
WDT_LOWER_LIMIT	WDT Lower Limit Register
WDT_UPPER_LIMIT	WDT Upper Limit Register
WDT_WARN_LIMIT	WDT Warn Limit Register
WDT_CONFIG	WDT Configuration Register
WDT_CNT	WDT Count Register
WDT_LOCK	WDT Lock Register
WDT_SERVICE	WDT Service Register
WDT_INTR	WDT Interrupt Register
WDT_INTR_SET	WDT Interrupt Set Register
WDT_INTR_MASK	WDT Interrupt Mask Register
WDT_INTR_MASKED	WDT Interrupt Masked Register
CRWDT	
CRWDT_CTL	Challenge Response Watchdog Control Register (Lock Protected)
CRWDT_CHALLENGE	Challenge Response Watchdog Challenge Value Register
CRWDT_RESPONSE	Challenge Response Watchdog Response Value Register
CRWDT_UPCNT	Challenge Response Watchdog Up Counter Register
CRWDT_EARLY	Challenge Response Watchdog Early Limit Register (Lock Protected)
CRWDT_WARN	Challenge Response Watchdog Warning Limit Register (Lock Protected)
CRWDT_LATE	Challenge Response Watchdog Late Limit Register (Lock Protected)
CRWDT_CONFIG	Challenge Response Watchdog Configuration Register
Lifetime Counter	
LIFETIME_CTL	Lifetime Counter Control Register (Lock Protected)
LIFETIME_WAKEUP	Lifetime Wakeup Value Register
LIFETIME_COUNTER	Lifetime Counter Current Value Register

16 Reset system and interrupts

The PSoC™ 4 HV PA can be reset from a variety of sources including a software reset. Reset events are asynchronous and guarantee reversion to a known state. The reset cause is recorded in a register (RES_CAUSE), which is sticky through reset and allows software to determine the cause of the reset. An XRES pin is reserved for external reset to avoid complications with configuration and multiple pin functions during power-on or reconfiguration. This pin is not used in normal applications, it is intended for test purpose and production, for example, in conjunction with debugging and flashing the device.

The following events cause resets:

- Power-on reset (POR) to hold the device in reset while the power supply ramps up
- Brownout reset (BOD) to reset the device if the power supply falls below specifications during operation
- Over-Voltage Detection (OVD) to reset the device if the power supply rises above specifications during operation
- Watchdog reset (RESET_WDT) to reset the device if firmware execution fails to service the watchdog timer
- Challenge/Response Watchdog reset (RESET_CRWDT) to reset the device if a failed response occurs
- Fault Infrastructure reset (RESET_ACT_FAULT) to reset the device if Fault Subsystem captured fault
- Software initiated reset (RESET_SOFT) to reset the device on demand using firmware
- External reset (XRES) to reset the device using an external electrical signal
- Protection fault reset (RESET_PROT_FAULT) to reset the device if unauthorized operating conditions occur

The interrupt status of SRSSHV is available in the SRSS_INTR register. The SRSS_INTR indicates those events generated by SRSS peripherals such as CRWDT, clock calibration counter, wakeup of lifetime counter, and regulator over-temp interrupt.

16.1 Reset sources

The following sections provide a description of the reset sources available in PSoC™ 4 HV PA.

16.1.1 Power-on reset

The Power-On-Reset (POR) circuits provide a reset pulse during the initial power ramp. POR circuits monitor V_{CCD} (core) voltage. The POR guarantees that all circuits are properly initialized before release. POR circuits are used during initial chip power-up and then disabled.

16.1.2 Brownout reset

The Brownout Reset (BOD) circuit protects the operating or retaining logic from possibly unsafe supply conditions by applying reset to the device. The BOD circuit monitors the V_{DDD} and V_{CCD} voltage. The BOD circuit generates a reset if voltage dips below the minimum safe operating voltage. The system will not come out of RESET until the supply is detected to be valid again.

To enable firmware to distinguish a normal power cycle from a brownout event, a special register is provided (RES_CAUSE), which will not be cleared after a BOD generated RESET. However, this register will be cleared if the device goes through POR or XRES.

16.1.3 Over-voltage detection

The Over-Voltage Detection (OVD) circuit protects the operating or retaining logic from possibly unsafe supply conditions by applying reset to the device. OVD circuit monitors the V_{DDD} and V_{CCD} voltage. As the name suggests, the OVD circuit maintains a device reset, if V_{CCD} or V_{DDD} supply stays higher than thresholds. The system will not come out of RESET until the supply is detected to be valid again.

The OVD circuit can generate a reset in all device power modes except POR or XRES. Note that the OVD settings are OFF by default. User software needs to enable this features by PWR_SSV_CTL register.

16.1.4 Watchdog reset

Watchdog reset (RESET_WDT) detects errant code by causing a reset if the watchdog timer is not cleared within the user-specified time limit.

The RESET_WDT status bit of the RES_CAUSE register is set when a watchdog reset occurs. This bit remains set until cleared or until a POR, XRES, or BOD reset; for example, in the case of a device power cycle. All other resets leave this bit untouched.

For more details, see the **"Watchdog timer"** on page 117.

16.1.5 Challenge/Response watchdog reset

Challenge/Response Watchdog reset (RESET_CRWDT) detects errant code by causing a reset when a failed response occurs. The RESET_CRWDT status bit of the RES_CAUSE register is set when a watchdog reset occurs. This bit remains set until cleared or until a POR, XRES, or BOD reset; for example, in the case of a device power cycle. All other resets leave this bit untouched.

For more details, see the **"Watchdog timer"** on page 117.

16.1.6 Fault infrastructure reset

Fault Infrastructure reset (RESET_ACT_FAULT) to reset the device if Fault Subsystem captured fault. This type of reset brings regular MMIO registers to their default/reset state. The RESET_ACT_FAULT status bit of the RES_CAUSE register is set when a fault Infrastructure reset occurs. This bit remains set until cleared or until a POR, XRES, or BOD reset; for example, in the case of a device power cycle. All other resets leave this bit untouched.

For more details, see the **"Fault subsystem"** on page 87.

16.1.7 Software initiated reset

Software initiated reset (RESET_SOFT) is a mechanism that allows a software-driven reset. The Cortex®-M0 application interrupt and reset control register (CM0_AIRCR) forces a device reset when a '1' is written into the SYSRESETREQ bit. CM0_AIRCR requires a value of A05F written to the top two bytes for writes. Therefore, write A05F0004 for the reset.

The RESET_SOFT status bit of the RES_CAUSE register is set when a software reset occurs. This bit remains set until cleared or until a POR, XRES, or BOD reset; for example, in the case of a device power cycle. All other resets leave this bit untouched.

16.1.8 External reset

External reset (XRES) is a user-supplied reset that causes immediate system reset when asserted. The XRES pin is active low – a high voltage on the pin has no effect and a low voltage causes a reset. The pin is pulled high inside the device. XRES is available as a dedicated pin in most of the devices. For detailed pinout, refer to the pinout section of the PSoC™ 4 HV PA datasheet.

The XRES pin holds the device in reset while held active. When the pin is released, the device goes through a normal boot sequence. The logical thresholds for XRES and other electrical characteristics, are listed in the Electrical Specifications section of the PSoC™ 4 HV PA datasheet.

XRES events do not set a reset cause status bit, but can be partially inferred by the absence of any other reset source. If no other reset event is detected, then the reset is caused by POR, BOD, or XRES.

16.1.9 Protection fault reset

Protection fault reset (RESET_PROT_FAULT) detects unauthorized protection violations and causes a device reset if they occur. One example of a protection fault is if a debug breakpoint is reached while executing privileged code. For details about privilege code, see **"Privileged"** on page 112.

The RESET_PROT_FAULT bit of the RES_CAUSE register is set when a protection fault occurs. This bit remains set until cleared or until a POR, XRES, or BOD reset; for example, in the case of a device power cycle. All other resets leave this bit untouched.

16.2 Reset levels

The PSoC™ 4 HV PA has three levels of reset:

- Cold, also known as High Voltage: Clears all registers except SRSS_RES_CAUSE. Triggered by XRES, POR, BOD, OVD, and WDT.
- Warm, also known as Low Voltage: Resets all registers unless specifically mentioned in the register description (fault system). Triggered by protection fault, fault system, CRWDT, or SYSRESETREQ.
- CPU reset: Resets only the M0+; no impact to any peripheral registers. Nothing is logged in the reset cause register. Triggered by VECTRESET or setting PC to start of application.

16.3 Identifying reset sources

When the device comes out of reset, it is often useful to know the cause of the most recent or even older resets. This is achieved in the device primarily through the RES_CAUSE register. This register has specific status bits allocated for some of the reset sources. The RES_CAUSE register supports detection of all above resets. The bits are set on the occurrence of the corresponding reset and remain set after the reset, until cleared or a loss of retention, such as a POR reset, external reset, or brownout detect.

Note: During the initial power ramp, VDD will be below the trip thresholds of both the POR and VDD BOD circuits. The reset cause register (RES_CAUSE) will always have the POR cause set after a power ramp. It is normal but not guaranteed that BOD_VDDD will also be included as a cause.

16.4 SRSS interrupts

The SRSS_INTR register indicates SRSSHV interrupts. The SRSS_INTR_MASK register decides whether the interrupts are forwarded to the CPU.The following are the interrupt sources available in SRSSHV.

16.4.1 CRWDT interrupt

The SRSS_INTR.CRWDT bit is set by warning and early faults of CRWDT when enabled by CRWDT_CONFIG. For more details, see the **"Watchdog timer"** on page 117.

16.4.2 Clock calibration counter interrupt

The SRSS_INTR.CLK_CAL bit is set when the clock calibration counter is done. This bit is reset during Deep Sleep mode. For more details, see the **"Clocking system"** on page 93.

16.4.3 Wakeup interrupt from lifetime counter

The SRSS_INTR.LIFETIME_WAKEUP bit is set when LIFETIME_COUNTER = LIFETIME_WAKEUP. This is a Deep Sleep interrupt. The interrupt source will reset only with hard reset (power related reset, XRES, WDT/CRWDT reset). For more details, see the **"Watchdog timer"** on page 117.

16.4.4 Over-temp interrupt

The SRSS_INTR.TEMP_HIGH bit is set when a short circuit exists on the V_{CCD} pin or when extreme loads are applied on the I/O cells causing the die to overheat. Firmware is encouraged to shut down all I/O cells and then go to Deep Sleep mode when this interrupt occurs to protect against such conditions.

16.5 Register list

Table 16-1. Reset system and interrupts register list

Register name	Description
CM0P_AIRCR	Cortex®-M0+ Application Interrupt and Reset Control Register - This register allows initiation of software resets, among other Cortex®-M0+ functions.
RES_CAUSE	Reset Cause Register - This register captures the cause of recent resets.
SRSS_INTR	This register shows interrupt requests from the SRSS peripheral.
SRSS_INTR_SET	This register is used for firmware testing.
SRSS_INTR_MASK	This register controls forwarding of the interrupt to CPU.
SRSS_MASKED	This register shows the logical AND of the corresponding SRSS interrupt request (SRSS Interrupt register) and mask bits (SRSS Interrupt Mask register)

17 Device security and register protection

PSoC™ 4 HV PA offers a number of options for protecting user designs from unauthorized access or copying and for protecting critical registers. Disabling debug features and enabling flash protection provide a high level of security.

The debug circuits are enabled by default and can only be disabled in firmware. If disabled, the only way to re-enable them is to erase the entire device, clear flash protection, and reprogram the device with new firmware that enables debugging. Additionally, all device interfaces can be permanently disabled for applications concerned about phishing attacks due to a maliciously reprogrammed device or attempts to defeat security by starting and interrupting flash programming sequences. Permanently disabling interfaces is not recommended for most applications because the designer cannot access the device. For more information, as well as a discussion on flash row and chip protection, see the PSoC™ 4 HV PA datasheet.

Note: Because all programming, debug, and test interfaces are disabled when maximum device security is enabled, PSoC™ 4 HV PA devices with full device security enabled may not be returned for failure analysis.

17.1 Device security

The PSoC™ 4 HV PA device security system has the following features:

- User-selectable levels of protection.
- In the most secure case provided, the chip can be “locked” such that it cannot be acquired for test/debug and it cannot enter erase cycles. Interrupting erase cycles is a known way for hackers to leave chips in an undefined state and open to observation.
- CPU execution in a privileged mode by use of the non-maskable interrupt (NMI). When in privileged mode, NMI remains asserted to prevent any inadvertent return from interrupt instructions causing a security leak.

In addition to these, the device offers protection for individual flash row data.

17.1.1 How it works

17.1.1.1 Device protection modes

The CPU operates in normal user mode or in privileged mode, and the device operates in one of four protection modes: BOOT, OPEN, PROTECTED, and KILL. Each mode provides specific capabilities for the CPU software and debug.

- **BOOT mode:** The device comes out of reset in BOOT mode. It stays there until its protection state is copied from supervisor flash to the protection control register. The debug-access port is stalled until this has happened. BOOT is a transitory mode required to set the part to its configured protection state. During BOOT mode, the CPU always operates in privileged mode.
- **OPEN mode:** This is the factory default. The CPU can operate in user mode or privileged mode. In user mode, flash can be programmed and debugger features are supported. In privileged mode, access restrictions are enforced.
- **PROTECTED mode:** The user may change the mode from OPEN to PROTECTED. This mode disables all debug access to user code or memory. In protected mode, only few registers are accessible; debug access to registers to reprogram flash is not available. The mode can be set back to OPEN but only after completely erasing the flash.
- **KILL mode:** The user may change the mode from OPEN to KILL. This mode disables all debug access to user code or memory. Access to most registers is still available by internal software; access to registers to reprogram/erase is available. The devices in KILL mode may not be returned for failure analysis.

For system calls details, see **“System calls”** on page 377.

17.1.1.2 Flash security

The PSoC™ 4 HV PA devices include a flexible flash-protection system that controls access to flash memory. This feature is designed to secure proprietary code, but it can also be used to protect against inadvertent writes to the bootloader portion of flash.

Flash memory is organized in rows. You can assign one of two protection levels to each row; see Table 17-1. Flash protection levels can only be changed by performing a complete flash erase.

For more details, see the **"Flash Memory"** on page 67 and **"Nonvolatile memory programming"** on page 374.

Table 17-1. Flash Protection Levels

Protection setting	Allowed	Not Allowed
Unprotected	External read and write, Internal read and write	–
Full Protection	External read[a)] Internal read	External write, Internal write

a) To protect the device from external read operations, you should change the device protection settings to PROTECTED.

17.2 Register protection

Register protection is for the critical registers. These critical registers are protected by means of a "magic key" written to the REG_PROT.MAGIC in SRSSHV register. After writing the correct key, users can update these critical registers. This is a non-retention register that resets in DEEPSLEEP power mode.

17.2.1 How it works

Setting the REG_PROT.MAGIC bits to the value 0xF08169E7 unlocks access to lock-protected registers. These protected registers cannot be written to unless this value is written into them. Writing a value other than the magic key will disable access to the registers. The register POR value is set to "unlock" the register access.

The following registers are locked:

- PWR_CONTROL
- PWR_KEY_DELAY
- CLK_SELECT
- CLK_ILO_CONFIG
- CLK_IMO_CONFIG
- CRWDT_CTL
- CRWDT_EARLY
- PWR_SSV_CTL
- CRWDT_WARN
- CRWDT_LATE
- LIFETIME_CTL
- HPOSC_CTL
- PILO_CTL
- All trim registers

The same register protection is available in PACSS. See **"Registers"** on page 338 for PACSS register protection.

17.3 Disable SWD in firmware

The PSoC™ 4 HV PA Program and Debug interface uses a serial wire debug (SWD) interface as the communication protocol with the external device. The SWD physical port pins (SWDIO and SWDCK) communicate with the Cortex®-M0+ debug and access port (DAP) through the high-speed I/O matrix (HSIOM). The default value of SWP ports are set to HSIOM_PORT_SEL0 = 0xE (See **Table 18-4**).

To disable the SWD in firmware, the HSIOM settings should change to 0x0; change the drive mode to weak pull-down as shown here:

- Disable SWDIO
  - HSIOM_PORT_SEL0.IO6_SEL=0
  - GPIO_PRT0_PC.DM6=7
  - GPIO_PRT0_PC2.INP_DIS6=1
  - GPIO_PRT0_DR.DATA6=0
- Disable SWDCK
  - HSIOM_PORT_SEL0.IO7_SEL=0
  - GPIO_PRT0_PC.DM7=7
  - GPIO_PRT0_PC2.INP_DIS7=1
  - GPIO_PRT0_DR.DATA7=0

17.4 Privileged registers

The PSoC™ 4 HV PA has some privileged registers in the MMIO space, FLASH, SRAM, and SROM memory. Privileged registers may not be read or written to by the Cortex-M0+ core or the DAP, unless they are in Privileged mode (See **"Chip operational modes"** on page 111).

Notes:

- Reading or writing protected memory locations may result in a protection violation error (See **"Fault subsystem"** on page 87).
- A protection violation may occur that requires a RESET. This includes hitting a debug breakpoint while in Privileged mode (RES_CAUSE.RESET_PROT_FAULT).
- The privileged registers are not listed in the *PSoC™ 4 HV PA Registers TRM*.

Section C: I/O System

This section encompasses the following chapter:

- **"I/O system"** on page 141

Top Level Architecture

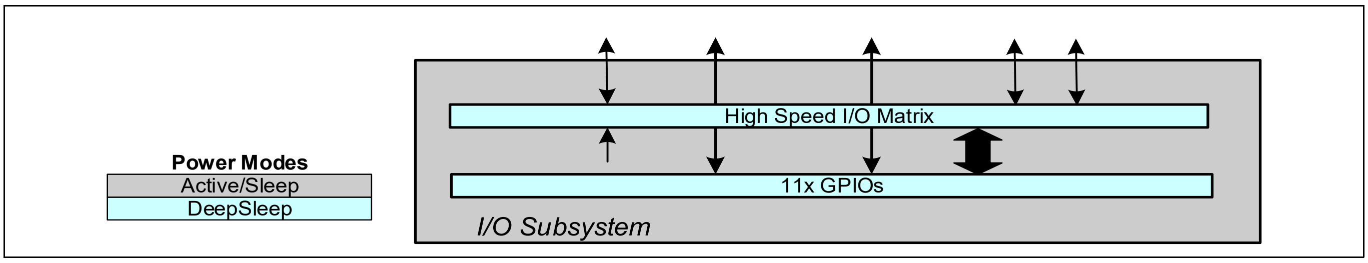


Figure 18-1. I/O System block diagram

18 I/O system

This chapter explains the PSoC™ 4 HV PA I/O system, its features, architecture, operating modes, and interrupts. The GPIO pins in PSoC™ 4 HV PA are grouped into ports; a port can have a maximum of eight GPIOs. The PSoC™ 4 HV PA device has eight GPIOs arranged in one port (P0.x), and three peripheral connection I/Os (P1.0 to P1.2) shared with the external temperature sensor pins (VTEMP_SUP/ VTEMP/ VTEMP_RET).

18.1 Features

The GPIOs have these features:

- Output drive modes include push-pull (strong or weak), open drain/source, high-z, and pull-up/-down
- Selectable CMOS and low-voltage LVTTL input buffer mode
- Edge-triggered interrupts on rising edge, falling edge, or on both the edges, on pin basis
- Individual control of input and output disables
- Hold mode for latching previous state (for retaining I/O state in Deep Sleep)
- Selectable slew rates allowing dV/dt control to assist with noise control to improve EMI

18.2 I/O cell architecture

Figure 18-2 shows the I/O cell architecture. It comprises of an input buffer and an output driver. This architecture is present in every GPIO cell. It connects to the HSIOM multiplexers for the digital input and the output signal.

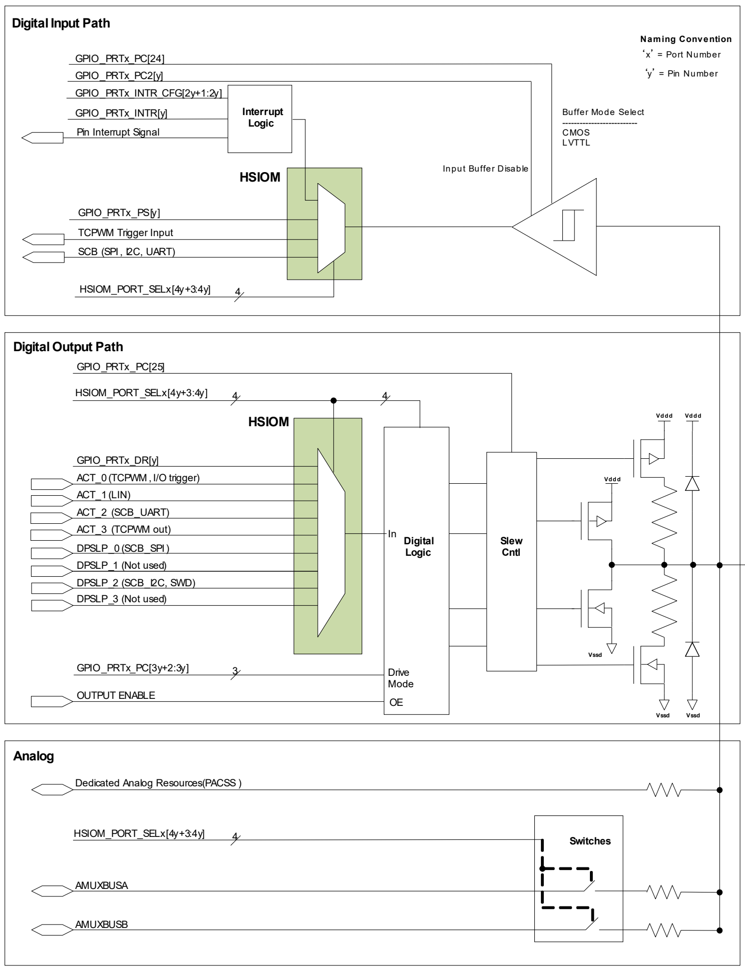


Figure 18-2. GPIO block diagram

18.2.1 Digital input buffer

The digital input buffer provides a high-impedance buffer for the external digital input. The buffer is enabled and disabled by the INP_DIS bit of the Port Configuration Register 2 (GPIO_PRTx_PC2, where x is the port number). The buffer is configurable for the following modes:

- CMOS
- LVTTL
- 1.8 V (Not used in PSoC™ 4 HV PA)

These buffer modes are selected by the PORT_VTRIP_SEL and PORT_IB_MODE_SEL bit (GPIO_PRTx_PC)of the Port Configuration register.

Table 18-1. Input buffer modes

PORT_IB_MODE_SEL	PORT_VTRIP_SEL	Input Buffer Mode
0b	0b	CMOS
0b	1b	LVTTL
1b	0b or 1b	1.8 V (Do not use this setting)

The threshold values for each mode can be obtained from the PSoC™ 4 HV PA datasheet. The output of the input buffer is connected to the HSIOM for routing to the selected peripherals. Writing to the HSIOM port select register (HSIOM_PORT_SELx) selects the peripheral. The digital input peripherals in the HSIOM, shown in **Figure 18-2**, are pin dependent. See the PSoC™ 4 HV PA datasheet to know the functions available for each pin.

18.2.2 Digital output driver

Pins are driven by the digital output driver. It consists of circuitry to implement different drive modes and slew rate control for the digital output signals. The peripheral connects to the digital output driver through the HSIOM; a particular peripheral is selected by writing to the HSIOM port select register (HSIOM_PORT_SELx).

In PSoC™ 4 HV PA I/Os are driven with V_{DDD} supply. Each GPIO pin has ESD diodes to clamp the pin voltage to the V_{DDD} source. Ensure that the voltage at the pin does not exceed the I/O supply voltage V_{DDD} and drop below V_{SSD}. For the absolute maximum and minimum GPIO voltage, see the device datasheet. The digital output driver can be enabled and disabled using the DSI signal from the peripheral or data register (GPIO_PRTx_DR) associated with the output pin. See **"High-speed I/O matrix"** on page 145 to know about the peripheral source selection for the data and to enable or disable control source selection.

18.2.3 Drive modes

Each I/O is individually configurable into one of eight drive modes using the Port Configuration register, GPIO_PRTx_PC. **Table 18-2** lists the drive modes. **Figure 18-3** is a simplified output driver diagram that shows the pin view based on each of the eight drive modes.

Table 18-2. Drive mode settings

GPIO_PRTx_PC ('x' denotes port number and 'y' denotes pin number)

Bits	Drive Mode	Value	Data = 1	Data = 0
3y+2: 3y	SEL'y'	Selects Drive Mode for Pin 'y' (0 ≤ y ≤ 7)		
	High-Impedance Analog	0	High Z	High Z
	High-impedance Digital	1	High Z	High Z
	Resistive Pull Up	2	Weak 1	Strong 0
	Resistive Pull Down	3	Strong 1	Weak 0
	Open Drain, Drives Low	4	High Z	Strong 0
	Open Drain, Drives High	5	Strong 1	High Z
	Strong Drive	6	Strong 1	Strong 0
	Resistive Pull Up and Down	7	Weak 1	Weak 0

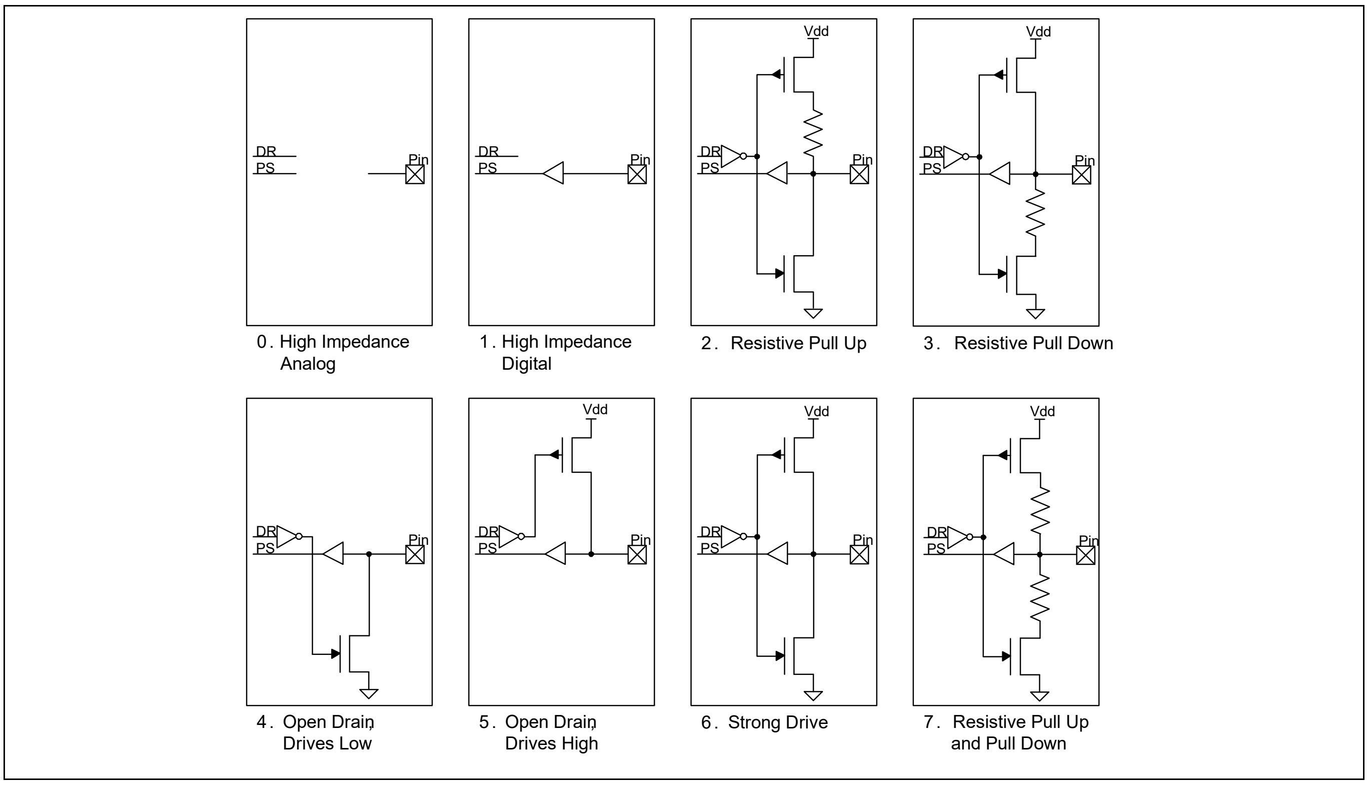


Figure 18-3. I/O Drive Mode block diagram

- High-Impedance Analog

High-impedance analog mode is the default reset state; both output driver and digital input buffer are turned off. This state prevents an external voltage from causing a current to flow into the digital input buffer. This drive mode is recommended for pins that are floating or that support an analog voltage. High-impedance analog pins cannot be used for digital inputs. Reading the pin state register returns a 0x00 regardless of the data register value. To achieve the lowest device current in low-power modes, unused GPIOs must be configured to the high-impedance analog mode.

- High-Impedance Digital

High-impedance digital mode is the standard high-impedance (High Z) state recommended for digital inputs. In this state, the input buffer is enabled for digital input signals.

- Resistive Pull-Up or Resistive Pull-Down

Resistive modes provide a series resistance in one of the data states and strong drive in the other. Pins can be used for either digital input or digital output in these modes. If resistive pull-up is required, a ‘1’ must be written to that pin’s Data Register bit. If resistive pull-down is required, a ‘0’ must be written to that pin’s Data Register. Interfacing mechanical switches is a common application of these drive modes. The resistive modes are also used to interface PSoC™ with open drain drive lines. Resistive pull-up is used when input is open drain low and resistive pull-down is used when input is open drain high.

- Open Drain Drives High and Open Drain Drives Low

Open drain modes provide high impedance in one of the data states and strong drive in the other. The pins can be used as digital input or output in these modes. Therefore, these modes are widely used in bi-directional digital communication. Open drain drive high mode is used when signal is externally pulled down and open drain drive low is used when signal is externally pulled high. A common application for open drain drives low mode is driving I^{2}C bus signal lines.

- Strong Drive

The strong drive mode is the standard digital output mode for pins; it provides a strong CMOS output drive in both high and low states. Strong drive mode pins must not be used as inputs under normal circumstances. This mode is often used for digital output signals or to drive external transistors.

- Resistive Pull-Up and Resistive Pull-Down

In the resistive pull-up and resistive pull-down mode, the GPIO will have a series resistance in both logic 1 and logic 0 output states. The high data state is pulled up while the low data state is pulled down. This mode is used when the bus is driven by other signals that may cause shorts.

18.2.3.1 Slew rate control

GPIO pins have fast and slow output slew rate options in strong drive mode; this is configured using PORT_SLOW bit of the Port Configuration register (GPIO_PRTx_PC[25]). Slew rate is individually configurable for each port. This bit is cleared by default and the port works in fast slew mode. This bit can be set if a slow slew rate is required. Slower slew rate results in reduced EMI and crosstalk; hence, the slow option is recommended for low-frequency signals or signals without strict timing constraints.

18.3 High-speed I/O matrix

The high-speed I/O matrix (HSIOM) is a group of high-speed switches that routes GPIOs to the peripherals inside the device. As the GPIOs are shared for multiple functions, HSIOM multiplexes the pin and connects to a particular peripheral selected by the user. PSoC™ 4 HV PA ports connect directly to the HSIOM.The HSIOM_PORT_SELx register is provided to select the peripheral. It is a 32-bit wide register available for each port, with each pin occupying four bits. This register provides up to 16 different options for a pin as listed in **Table 18-3**.

Table 18-3. PSoC™ 4 HV PA HSIOM Port settings

HSIOM_PORT_SELx ('x' denotes port number and 'y' denotes pin number)			
Bits	**Name (SEL'y')**	**Value**	**Description (Selects pin 'y' source ($0 \leq y \leq 7$))**
4y+3 : 4y	DR	0x0	Pin is regular firmware-controlled I/O or connected to dedicated hardware block.
	–	0x1–0x5	Not used
	AMUXA	0x6	Pin is connected to AMUXBUS-A.
	AMUXB	0x7	Pin is connected to AMUXBUS-B.
	ACTIVE_0	0x8	ACT#0. Pin-specific Active source #0 (TCPWM, I/O trigger).
	ACTIVE_1	0x9	ACT#1. Pin-specific Active source #1 (LIN).
	ACTIVE_2	0xA	ACT#2. Pin-specific Active source #2 (SCB-UART).
	ACTIVE_3	0xB	ACT#3. Pin-specific Active source #3 (TCPWM out).
	DEEP_SLEEP_0	0xC	DP#0. Pin-specific Deep Sleep source #0 (SCB-SPI).
	DEEP_SLEEP_1	0xD	DP#1. Not used
	DEEP_SLEEP_2	0xE	DP#2. Pin-specific Deep Sleep source #2 (SCB-I^{2}C, SWD).
	DEEP_SLEEP_3	0xF	DP#3. Not used

Note The Active and Deep Sleep sources are pin dependent. See the "Pinouts" section of the device datasheet or more details on the features supported by each pin.

Each port pin can be assigned to one of multiple functions; it can, for example, be an analog I/O or a digital peripheral function. The pin assignments are shown in **Table 18-4**.

Table 18-4. Alternate Pin functions

Name	**Bits**	**I/O pad route name**										
		DR 0x0	**AMUXA 0x6**	**AMUXB 0x7**	**ACT#0 0x8**	**ACT#1 0x9**	**ACT#2 0xA**	**ACT#3 0xB**	**DS#0 0xC**	**DS#1 0xD**	**DS#2 0xE**	**DS#3 0xF**
HSIOM_PORT_SEL0 register												
P0.0	[3:0]	GPIO	AMUXA	AMUXB	tcpwm.tr_in[0]	lin.lin_rx[0]	scb.uart_rx:1	tcpwm.line[0]	scb.spi_clk		scb.i2c_scl	
P0.1	[7:4]	GPIO	AMUXA	AMUXB	tcpwm.tr_in[1]	lin.lin_tx[0]	scb.uart_tx:1	tcpwm.line_compl[0]	scb.spi_mosi		scb.i2c_sda	
P0.2	[11:8]	GPIO	AMUXA	AMUXB	tcpwm.tr_in[2]	lin.lin_en[0]		tcpwm.line[1]	scb.spi_miso			
P0.3	[15:12]	GPIO	AMUXA	AMUXB	tcpwm.tr_in[3]			tcpwm.line_compl[1]	scb.spi_select0			
P0.4	[19:16]	GPIO	AMUXA	AMUXB	peri.virt_in_0			tcpwm.line[2]	scb.spi_select1		cpuss.fault_out[0]	
P0.5	[23:20]	GPIO	AMUXA	AMUXB	srss.ext_clk			tcpwm.line_compl[2]	scb.spi_select2		cpuss.fault_out[1]	
P0.6	[27:24]	GPIO	AMUXA	AMUXB	peri.virt_in_1				scb.spi_select3		cpuss.swd_data	
P0.7	[31:28]	GPIO	AMUXA	AMUXB	peri.virt_in_2						cpuss.swd_clk	

Table 18-4. Alternate Pin functions (continued)

Name	Bits	I/O pad route name										
		DR 0x0	**AMUXA 0x6**	**AMUXB 0x7**	**ACT#0 0x8**	**ACT#1 0x9**	**ACT#2 0xA**	**ACT#3 0xB**	**DS#0 0xC**	**DS#1 0xD**	**DS#2 0xE**	**DS#3 0xF**
HSIOM_PORT_SEL1 register												
VTEMP_ SUP (P1.0)	[3:0]	GPIO / VTEMP_SUP[a)]	AMUXA	AMUXB		hvss.lin_ alt_rxd		tcpwm.line[3]				
VTEMP (P1.1)	[7:4]	GPIO / VTEMP[a]	AMUXA	AMUXB		hvss.lin_ alt_txd						
VTEMP_ RET (P1.2)	[11:8]	GPIO / VTEMP_RET[a]	AMUXA	AMUXB	peri.virt_in_3	hvss.lin_ alt_en		tcpwm.line_ compl[3]				

a) There is no peripheral logic for operating the external temp sensor function. The CPU should directly control VTEMP_SUP and VTEMP_RET using GPIO_PRTx_DR register (after configuring HSIOM_PORT_SEL1 for these I/Os to DR: 0x00).

18.4 I/O State on power up

During power up all the GPIOs are in high-impedance analog state and the input buffers are disabled. During run time, GPIOs can be configured by writing to the associated registers. Note that the pins supporting debug access port (DAP) connections (SWD lines) are always enabled as SWD lines during power up. However, the DAP connection can be disabled or reconfigured for general-purpose use through HSIOM. However, this reconfiguration takes place only after the device boots and start executing code.

18.5 Behavior in low-power modes

Table 18-5 shows the status of GPIOs in low-power modes.

Table 18-5. GPIO in Low-Power Modes

Low-Power Mode	Status
Sleep	• GPIOs are active and can be driven by peripherals such as TCPWM and SCBs, which can work in sleep mode. • Input buffers are active; thus an interrupt on any I/O can be used to wake up the CPU. • AMUXBUS connections are available.
Deep Sleep	• Active connections (ACT#n in **Table 18-4**) are in latched and hold state, deep-sleep connections (DS#n in **Table 18-4**) remain available. The SCB (I^2C and SPI) block can work in the Deep Sleep mode and can wake up the CPU on address match or SPI slave select event. • Input buffers are also active in this mode; pin interrupts are functional. • AMUXBUS connections are not available.

18.6 Interrupt

In the PSoC™ 4 HV PA device, all the port pins have the capability to generate interrupts. As shown in **Figure 18-3**, the pin signal is routed to the interrupt controller through the GPIO Edge Detect block.

Figure 18-4 shows the GPIO Edge Detect block architecture.

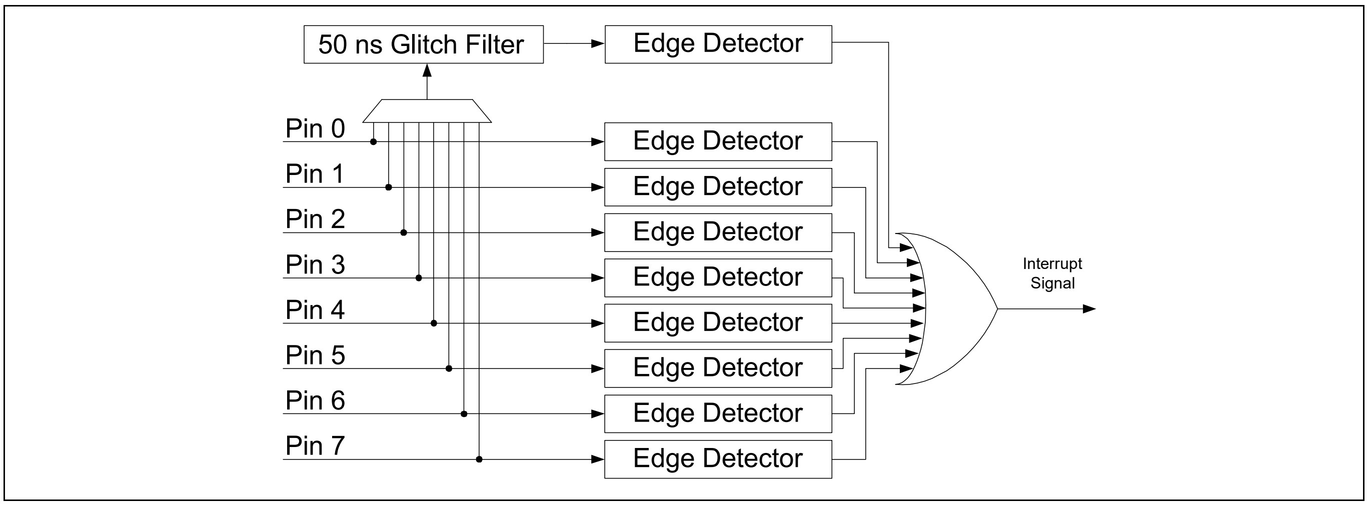


Figure 18-4. GPIO Edge Detect block architecture

An edge detector is present at each pin. It is capable of detecting rising edge, falling edge, and both edges without reconfiguration. The edge detector is configured by writing into the EDGE_SEL bits of the Port Interrupt Configuration register, GPIO_PRTx_INTR_CFG, as shown in **Table 18-6**.

Table 18-6. Edge Detector configuration

EDGE_SEL	Configuration
00	Interrupt is disabled
01	Interrupt on Rising Edge
10	Interrupt on Falling Edge
11	Interrupt on Both Edges

Besides the pins, edge detector is also present at the glitch filter output. This filter can be used on one of the pins of a port. The pin is selected by writing to the FLT_SEL field of the GPIO_PRTx_INTR_CFG register as shown in **Table 18-7**.

Table 18-7. Glitch filter input selection

FLT_SEL	Selected Pin
000	Pin 0 is selected
001	Pin 1 is selected
010	Pin 2 is selected
011	Pin 3 is selected
100	Pin 4 is selected
101	Pin 5 is selected
110	Pin 6 is selected
111	Pin 7 is selected

The edge detector outputs of a port are ORed together and then routed to the interrupt controller (NVIC in the CPU subsystem). Thus, there is only one interrupt vector per port. On a pin interrupt, it is required to know which pin caused an interrupt. This is done by reading the Port Interrupt Status register, GPIO_PRTx_INTR. This register not only includes the information on which pin triggered the interrupt, it also includes the pin status; it allows the CPU to read both information in a single read operation. This register has one more important use – to clear the interrupt. Writing '1' to the corresponding status bit clears the pin interrupt. It is important to clear the interrupt status bit; otherwise, the interrupt will occur repeatedly for a single trigger or respond only once for multiple triggers, which is explained later in this section. Also, note that when the Port Interrupt Control Status register is read when an interrupt is occurring on the corresponding port, it can result in the interrupt not being properly detected. Therefore, when using GPIO interrupts, it is recommended to read the status register only inside the corresponding interrupt service routine and not in any other part of the code. **Table 18-8** shows the Port Interrupt Status register bit fields.

Table 18-8. Port Interrupt Status Register

GPIO_PRTx_INTR	Description
0000b to 0111b	Interrupt status on pin 0 to pin 7. Writing '1' to the corresponding bit clears the interrupt
1000b	Interrupt status from the glitch filter
10000b to 10111	Pin 0 to Pin 7 status
11000b	Glitch filter output status

The edge detector block output is routed to the Interrupt Source Multiplexer shown in **Figure 7-3**, which gives an option of Level and Rising Edge detect. If the Level option is selected, an interrupt is triggered repeatedly as long as the Port Interrupt Status register bit is set. If the Rising Edge detect option is selected, an interrupt is triggered only once if the Port Interrupt Status register is not cleared. Thus, it is important to clear the interrupt status bit if the Edge Detect block is used.

18.7 Peripheral connections

18.7.1 Firmware controlled GPIO

See **Table 18-3** to know the HSIOM settings for a firmware controlled GPIO. GPIO_PRTx_DR is the data register used to read and write the output data for the GPIOs. A write operation to this register changes the GPIO output to the written value. Note that a read operation reflects the output data written to this register and not the current state of the GPIOs. Using this register, read-modify-write sequences can be safely performed on a port that has both input and output GPIOs.

In addition to the data register, three other registers – GPIO_PRTx_DR_SET, GPIO_PRTx_DR_CLR, and GPIO_PRTx_INV – are provided to set, clear, and invert the output data respectively of a specific pin in a port without affecting other pins. Writing '1' into these registers will set, clear, or invert; writing '0' will have no affect on the pin status.

GPIO_PRTx_PS is the I/O pad register that provides the state of the GPIOs when read. Writes to this register have no effect.

18.7.2 Analog I/O

Analog resources, such as the PACSS, which require low-impedance routing paths with hard-wired connections to pins. These analog connections are always present and are generally enabled by the destination analog circuitry. See the device datasheet for details on these dedicated pins. Some analog connections use dedicated analog pads (RS*), others use hard-wired connections to GPIOs (VTEMP*). When using a hard-wired analog connection to a GPIO, that GPIO should be configured in high-impedance analog mode (see **Table 18-2**).

GPIOs also feature connections to a two wire global analog bus called AMUXBUSA and AMUXBUSB. The GPIO should be configured in high-impedance analog mode and then routed to AMUXBUS using the HSIOM_PORT_SELx register.

18.7.3 Serial communication block (SCB)

These blocks are connected through HSIOM; there are no dedicated connections (see **Table 18-3** for HSIOM port settings). There is limited connectivity as determined by connections to HSIOM and documented in the Alternate Pin Functions section of the device datasheet. When the SPI mode is used, the SCB controls the digital output buffer drive mode for the input pin to keep the pin in the high-impedance state. This functionality overrides the drive mode settings, which is done using the GPIO_PRTx_PC register.

Refer to the **"Serial communications block (SCB)"** on page 152 for detailed functionalities.

18.7.4 Timer, counter, and pulse width modulator (TCPWM) block

These blocks are connected through HSIOM; there are no dedicated connections (See **Table 18-3**). There is limited connectivity as determined by connections to HSIOM and documented in the Alternate Pin Functions section of the PSoC™ 4 HV PA datasheet. Note that when the TCPWM block inputs such as start and stop are taken from the pins, the drive mode can be only high-Z digital because the TCPWM block disables the output buffer at the input pins.

Refer to the **"Timer, Counter, and PWM"** on page 244 for detailed functionalities.

18.7.5 LIN controller (MXLIN)

These blocks are connected through HSIOM; there are no dedicated connections (See **Table 18-3**). There is limited connectivity as determined by connections to HSIOM and documented in the Alternate Pin Functions section of the PSoC™ 4 HV PA datasheet. The MXLIN provides two LIN channels (MXLIN channel 0/1) that both connect to the HSIOM.

Refer to the **"Local interconnect network (LIN)"** on page 207 for detailed functionalities.

18.7.6 LIN PHY

The LIN PHY in the High-Voltage Subsystem (HVSS) connects to the HSIOM through the auxiliary interface, which makes the LIN PHY look like a set of three GPIO pads (P1.0 to P1.2). These pads are shared with the external temperature sensor pins (VTEMP/VTEMP_RET/VTEMP_SUP). The HVSS implements an alternate interface, which allows the PHY to be connected directly to GPIO pads so external signals can directly drive and receive data through the HVSS LIN PHY connected to the LIN bus. Note that these connections from the VTEMP pins to the LIN PHY are provided to enable certification of the LIN PHY. They are not expected to be used during normal operation. See **Table 18-5** for detailed settings.

Refer to the **"High-voltage subsystem"** on page 345 for detailed LIN PHY functionalities.

18.8 Registers

Table 18-9. I/O Registers

Name	Description
GPIO_INTR_CAUSE	Interrupt Port Cause Register
GPIO_DFT_IO_TEST	I/O SELF TEST Control Register for DFT purposes only
GPIO_PRTx_DR	Port Output Data Register
GPIO_PRTx_DR_SET	Port Output Data Set Register
GPIO_PRTx_DR_CLR	Port Output Data Clear Register
GPIO_PRTx_DR_INV	Port Output Data Inverting Register
GPIO_PRTx_PS	Port Pin State Register - Reads the logical pin state of I/O
GPIO_PRTx_PC	Port Configuration Register - Configures the output drive mode, input threshold, and slew rate
GPIO_PRTx_PC2	Port Secondary Configuration Register - Configures the input buffer of I/O pin
GPIO_PRTx_INTR_CFG	Port Interrupt Configuration Register
GPIO_PRTx_INTR	Port Interrupt Status Register
HSIOM_PORT_SELx	HSIOM Port Selection Register

Section D: Digital System

This section encompasses the following chapters:

- **"Serial communications block (SCB)"** on page 152
- **"Local interconnect network (LIN)"** on page 207
- **"Timer, Counter, and PWM"** on page 244

Top Level Architecture

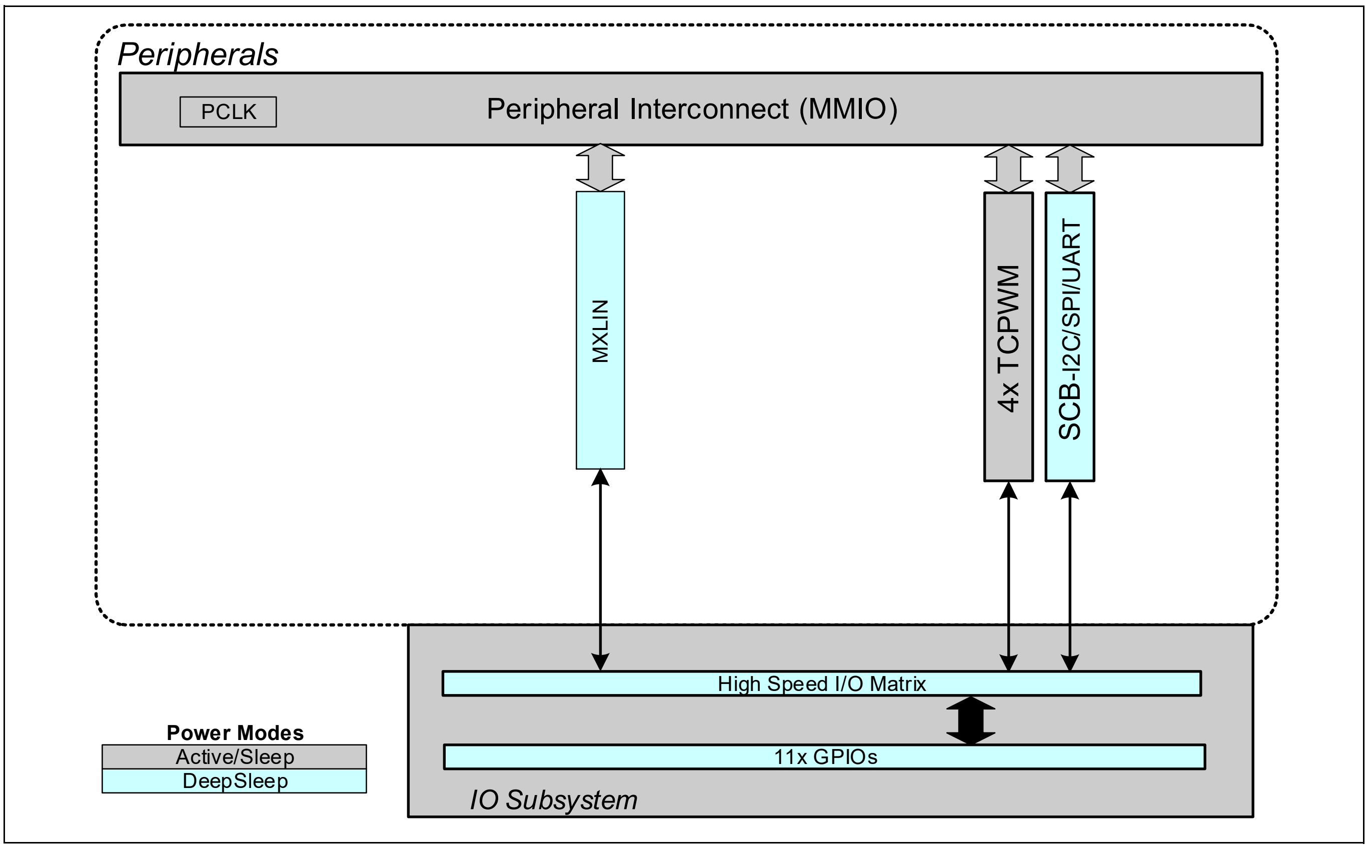


Figure 19-5. Digital System block diagram

19 Serial communications block (SCB)

The Serial Communications Block (SCB) of PSoC™ 4 HV PA supports three serial interface protocols: SPI, I^{2}C, and UART. Only one of the protocols is supported by an SCB at any given time. The PSoC™ 4 HV PA device has one SCB.

19.1 Features

This block supports the following features:

- Standard SPI master and slave functionality with Motorola, Texas Instruments, and National Semiconductor protocols
- Standard UART functionality with SmartCard reader, Local Interconnect Network (LIN), and IrDA protocols
- Standard I^{2}C master and slave functionality
- Standard LIN slave functionality with LIN v1.3 and LIN v2.1/2.2 specification compliance
- EZ mode for SPI and I^{2}C, which allows for operation without CPU intervention
- Low-power (Deep Sleep) mode of operation for SPI and I^{2}C protocols (using external clocking)

Each of the three protocols is explained in the following sections.

19.2 Serial peripheral interface (SPI)

The SPI protocol is a synchronous serial interface protocol. Devices operate in either master or slave mode. The master initiates the data transfer. The SCB supports single-master-multiple-slaves topology for SPI. Multiple slaves are supported with individual slave select lines.

You can use the SPI master mode when the PSoC™ has to communicate with one or more SPI slave devices. The SPI slave mode can be used when the PSoC™ has to communicate with an SPI master device.

19.2.1 Features

- Supports master and slave functionality
- Supports three types of SPI protocols:
  - Motorola SPI – modes 0, 1, 2, and 3
  - Texas Instruments SPI, with coinciding and preceding data frame indicator for mode 1
  - National Semiconductor (MicroWire) SPI for mode 0
- Supports up to four slave select lines
- Data frame size programmable from 4 bits to 16 bits
- Interrupts or polling CPU interface
- Programmable oversampling
- Supports EZ mode of operation (**"Easy SPI protocol"** on page 160)
  - EZSPI mode allows for operation without CPU intervention
- Supports externally clocked slave operation:
  - In this mode, the slave operates in Active, Sleep, and Deep Sleep system power modes

19.2.2 General description

Figure 19-6 illustrates an example of SPI master with four slaves.

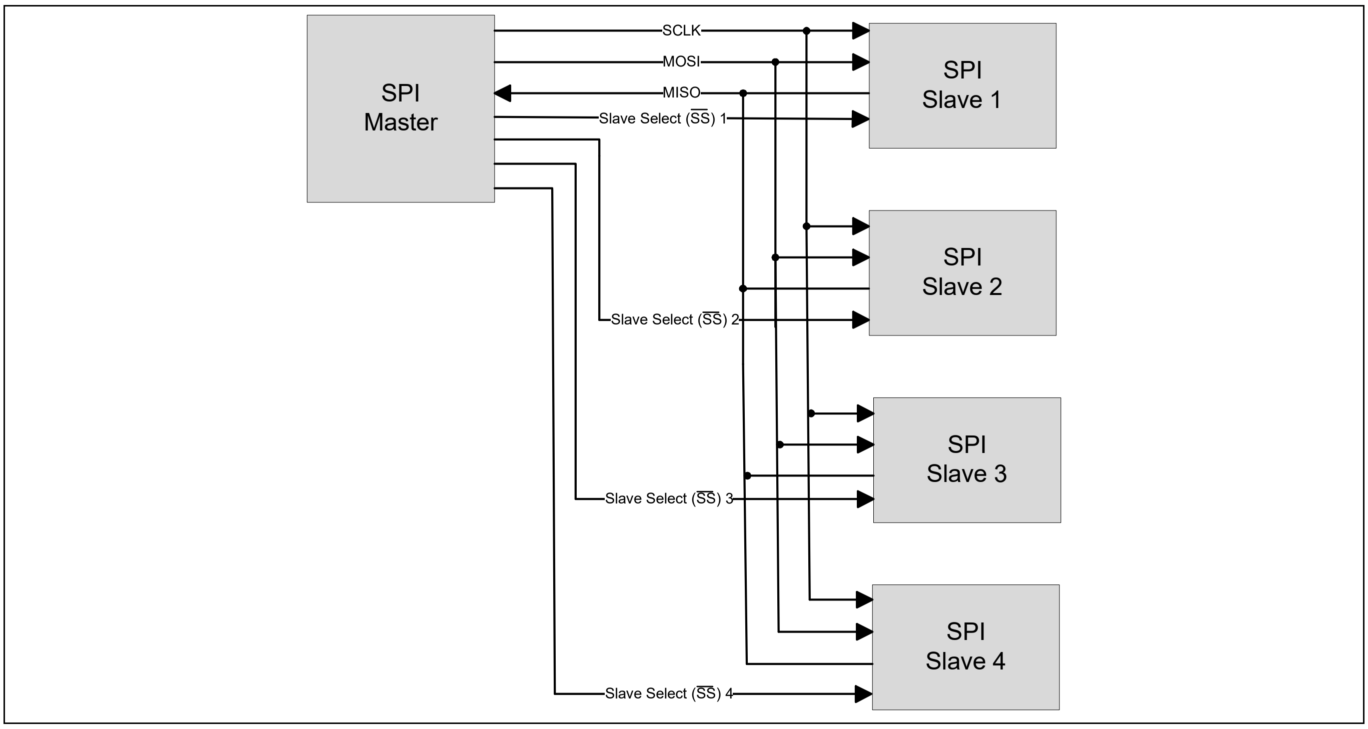

SPI Master
SCLK
MOSI
MISO
Slave Select ($\overline{SS}$) 1
Slave Select ($\overline{SS}$) 2
Slave Select ($\overline{SS}$) 3
Slave Select ($\overline{SS}$) 4
SPI Slave 1
SPI Slave 2
SPI Slave 3
SPI Slave 4

Figure 19-6. SPI example

A standard SPI interface consists of four signals as follows.

- SCLK: Serial clock (clock output from the master, input to the slave).
- MOSI: Master-out-slave-in (data output from the master, input to the slave).
- MISO: Master-in-slave-out (data input to the master, output from the slave).
- Slave Select ($\overline{SS}$): Typically an active low signal (output from the master, input to the slave).

A simple SPI data transfer involves the following: the master selects a slave by driving its $\overline{SS}$ line, then it drives data on the MOSI line and a clock on the SCLK line. The slave uses either of the edges of SCLK depending on the configuration to capture the data on the MOSI line; it also drives data on the MISO line, which is captured by the master.

By default, the SPI interface supports a data frame size of eight bits (1 byte). The data frame size can be configured to any value in the range 4 to 16 bits. The serial data can be transmitted either most significant bit (MSb) first or least significant bit (LSB) first.

Three different variants of the SPI protocol are supported by the SCB:

- Motorola SPI: This is the original SPI protocol.
- Texas Instruments SPI: A variation of the original SPI protocol, in which data frames are identified by a pulse on the $\overline{SS}$ line.
- National Semiconductors SPI: A half duplex variation of the original SPI protocol.

19.2.3 SPI modes of operation

19.2.3.1 Motorola SPI

The original SPI protocol was defined by Motorola. It is a full duplex protocol. Multiple data transfers may happen with the $\overline{\text{SS}}$ line held at ‘0’. As a result, slave devices must keep track of the progress of data transfers to separate individual data frames. When not transmitting data, the $\overline{\text{SS}}$ line is held at ‘1’ and SCLK is typically pulled low.

19.2.3.1.1 Modes of Motorola SPI

The Motorola SPI protocol has four different modes based on how data is driven and captured on the MOSI and MISO lines. These modes are determined by clock polarity (CPOL) and clock phase (CPHA).

Clock polarity determines the value of the SCLK line when not transmitting data. CPOL = '0' indicates that SCLK is ‘0’ when not transmitting data. CPOL = ‘1’ indicates that SCLK is ‘1’ when not transmitting data.

Clock phase determines when data is driven and captured. CPHA = 0 means sample (capture data) on the leading (first) clock edge, while CPHA = 1 means sample on the trailing (second) clock edge, regardless of whether that clock edge is rising or falling. With CPHA = 0, the data must be stable for setup time before the first clock cycle.

- Mode 0: CPOL is ‘0’, CPHA is ‘0’: Data is driven on a falling edge of SCLK. Data is captured on a rising edge of SCLK.
- Mode 1; CPOL is ‘0’, CPHA is ‘1’: Data is driven on a rising edge of SCLK. Data is captured on a falling edge of SCLK.
- Mode 2: CPOL is ‘1’, CPHA is ‘0’: Data is driven on a rising edge of SCLK. Data is captured on a falling edge of SCLK.
- Mode 3: CPOL is ‘1’, CPHA is ‘1’: Data is driven on a falling edge of SCLK. Data is captured on a rising edge of SCLK.

Figure 19-7 illustrates driving and capturing of MOSI/MISO data as a function of CPOL and CPHA.

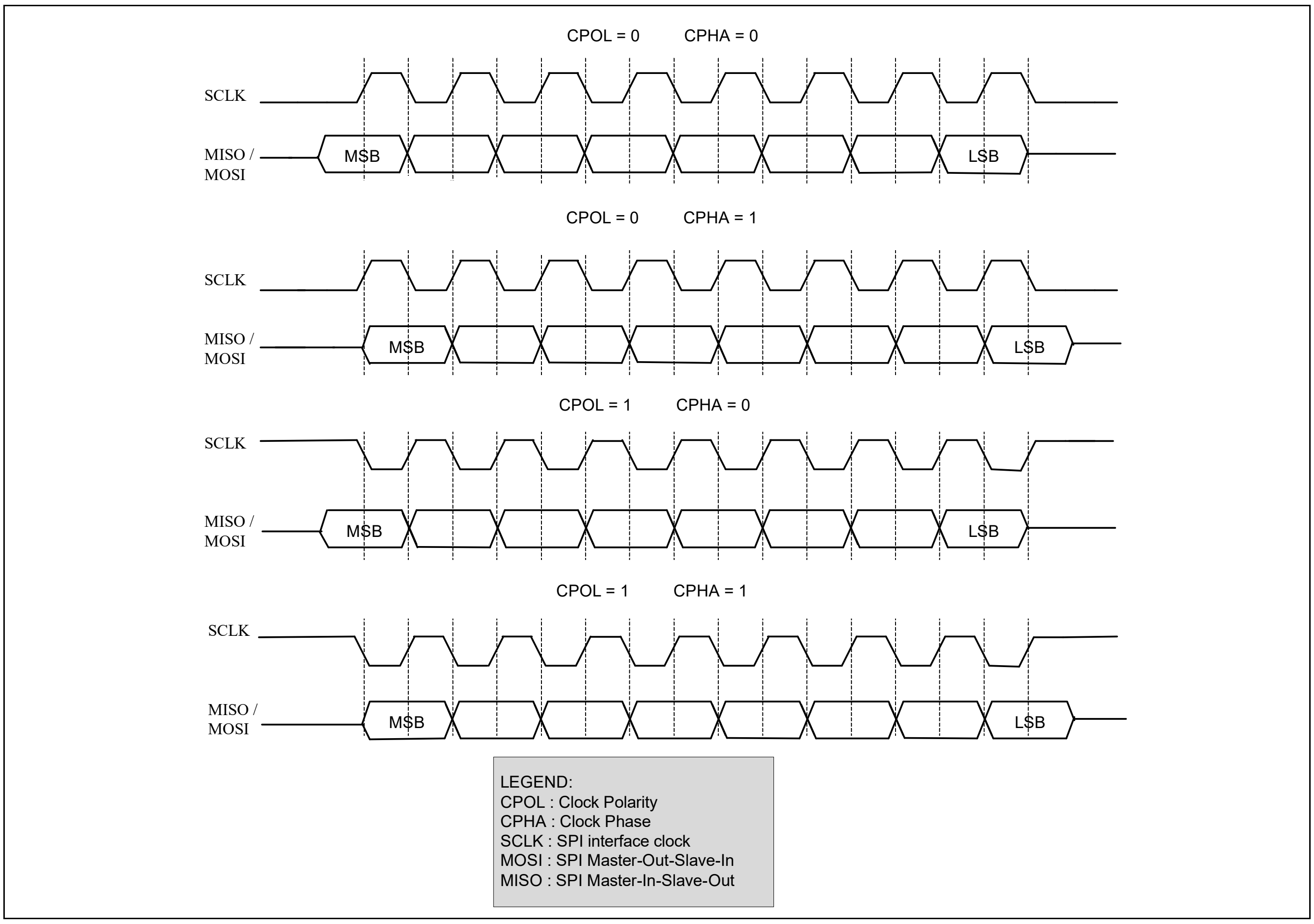


Figure 19-7. SPI Motorola, 4 Modes

Figure 19-8 illustrates a single 8-bit data transfer and two successive 8-bit data transfers in mode 0 (CPOL is ‘0’, CPHA is ‘0’).

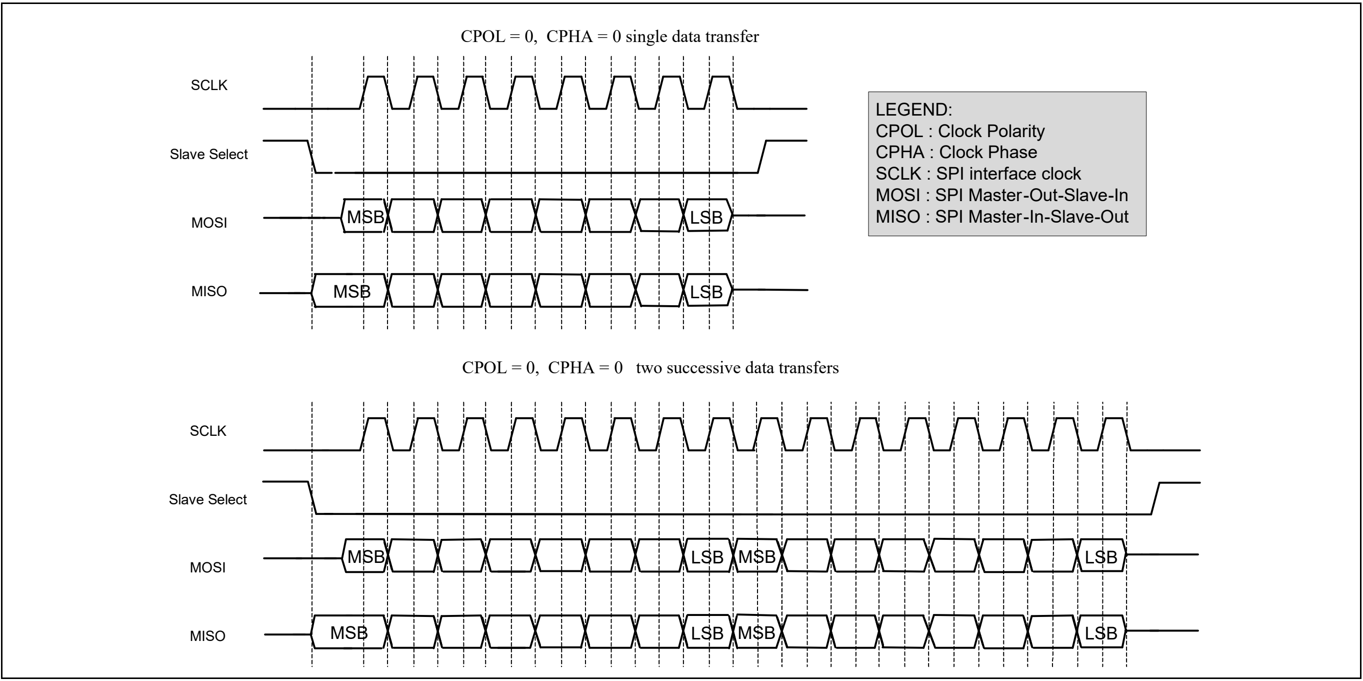

CPOL = 0, CPHA = 0 single data transfer
SCLK
Slave Select
MOSI
MISO
MSB
LSB
LEGEND:
CPOL : Clock Polarity
CPHA : Clock Phase
SCLK : SPI interface clock
MOSI : SPI Master-Out-Slave-In
MISO : SPI Master-In-Slave-Out
CPOL = 0, CPHA = 0 two successive data transfers
SCLK
Slave Select
MOSI
MISO
MSB
LSB

Figure 19-8. SPI Motorola data transfer example

19.2.3.1.2 Configuring SCB for SPI Motorola Mode

To configure the SCB for SPI Motorola mode, set various register bits in the following order:

1. Select SPI by writing ‘01’ to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Select SPI Motorola mode by writing ‘00’ to the MODE (bits [25:24]) of the SCB_SPI_CTRL register.
3. Select the mode of operation in Motorola by writing to the CPHA and CPOL fields (bits 2 and 3 respectively) of the SCB_SPI_CTRL register.
4. Follow steps 2 to 4 mentioned in **“Enabling and initializing SPI”** on page 164.

19.2.3.2 Texas Instruments SPI

The Texas Instruments' SPI protocol redefines the use of the $\overline{\text{SS}}$ signal. It uses the signal to indicate the start of a data transfer, rather than a low active slave select signal, as in the case of Motorola SPI. As a result, slave devices need not keep track of the progress of data transfers to separate individual data frames. The start of a transfer is indicated by a high active pulse of a single bit transfer period. This pulse may occur one cycle before the transmission of the first data bit, or may coincide with the transmission of the first data bit. The TI SPI protocol supports only mode 1 (CPOL is '0' and CPHA is '1'): data is driven on a rising edge of SCLK and data is captured on a falling edge of SCLK.

Figure 19-9 illustrates a single 8-bit data transfer and two successive 8-bit data transfers. The SELECT pulse precedes the first data bit. Note how the SELECT pulse of the second data transfer coincides with the last data bit of the first data transfer.

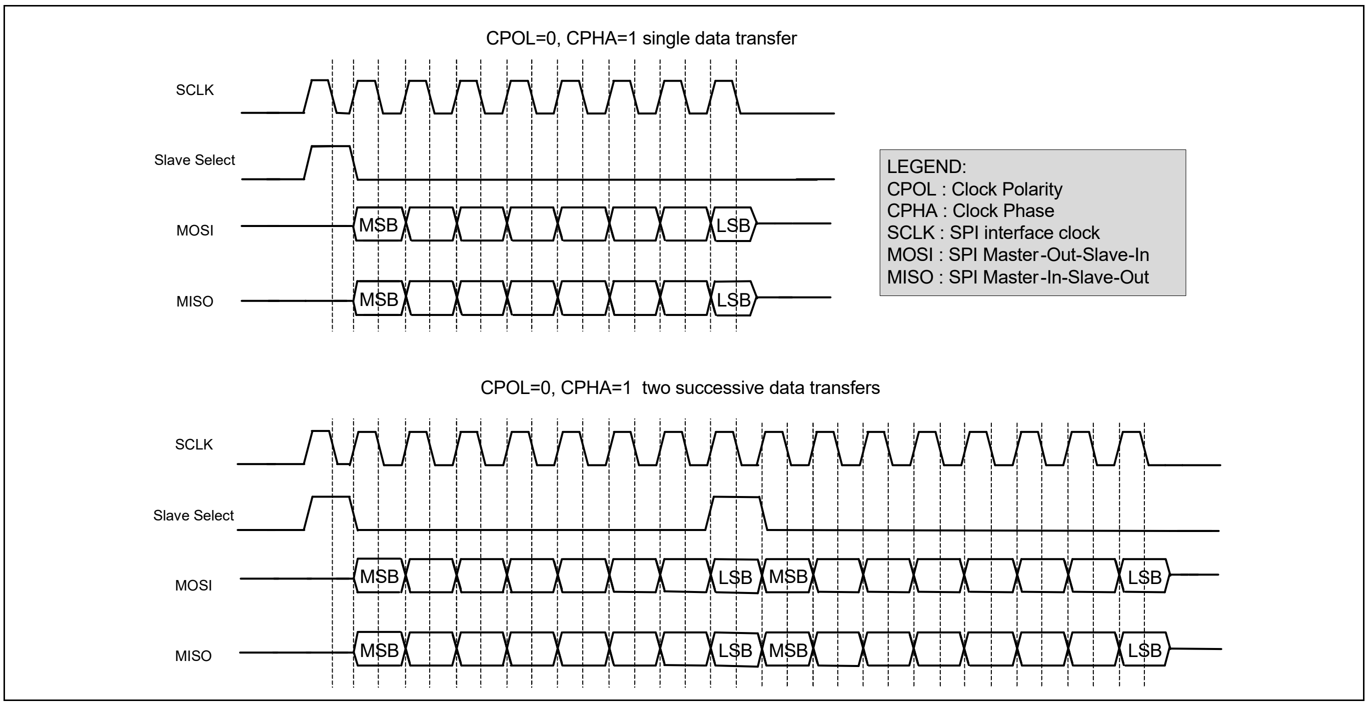

CPOL=0, CPHA=1 single data transfer
SCLK
Slave Select
MOSI
MISO
MSB
LSB
LEGEND:
CPOL : Clock Polarity
CPHA : Clock Phase
SCLK : SPI interface clock
MOSI : SPI Master-Out-Slave-In
MISO : SPI Master-In-Slave-Out
CPOL=0, CPHA=1 two successive data transfers
SCLK
Slave Select
MOSI
MISO
MSB
LSB
MSB
LSB

Figure 19-9. SPI TI data transfer example

Figure 19-10 illustrates a single 8-bit data transfer and two successive 8-bit data transfers. The SELECT pulse coincides with the first data bit of a frame.

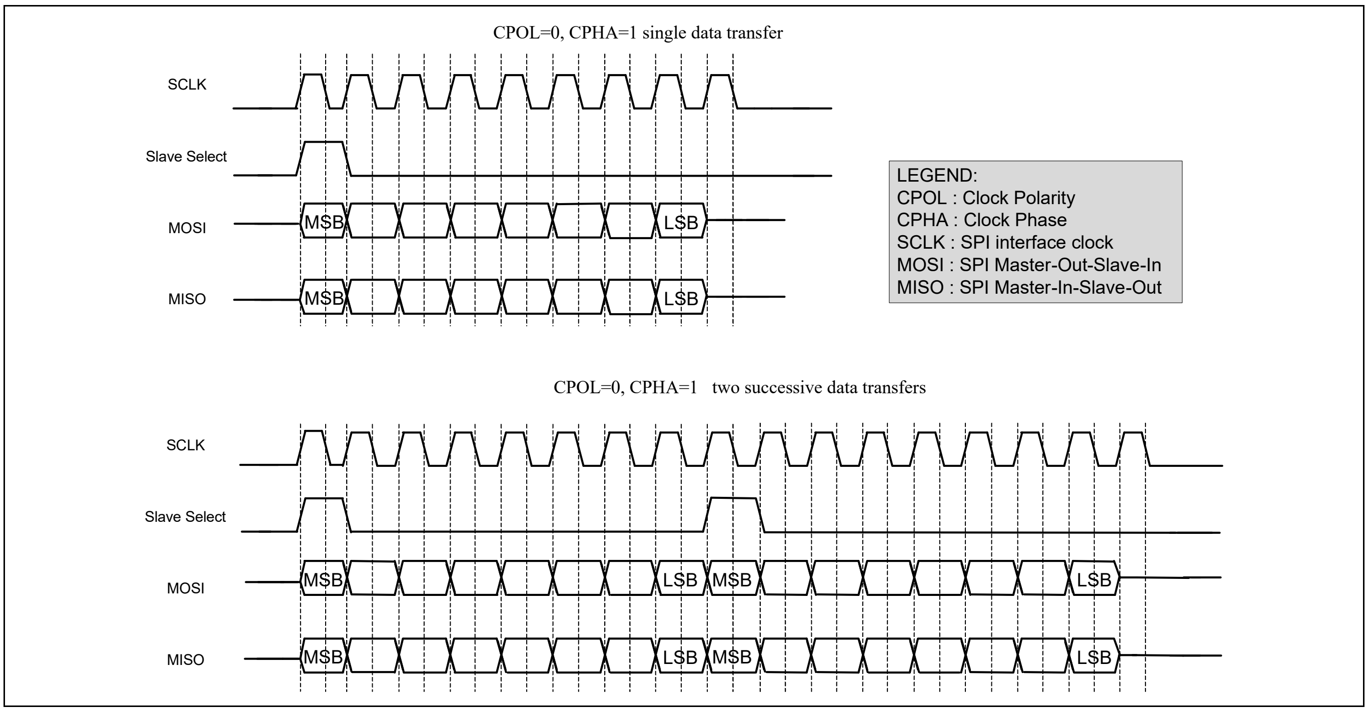

CPOL=0, CPHA=1 single data transfer
SCLK
Slave Select
MOSI
MISO
MSB
LSB
LEGEND:
CPOL : Clock Polarity
CPHA : Clock Phase
SCLK : SPI interface clock
MOSI : SPI Master-Out-Slave-In
MISO : SPI Master-In-Slave-Out
CPOL=0, CPHA=1 two successive data transfers

Figure 19-10. SPI TI data transfer example

19.2.3.2.1 Configuring SCB for SPI TI Mode

To configure the SCB for SPI TI mode, set various register bits in the following order:

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Select SPI TI mode by writing '01' to the MODE (bits [25:24]) of the SCB_SPI_CTRL register.
3. Select the mode of operation in TI by writing to the SELECT_PRECEDE field (bit 1) of the SCB_SPI_CTRL register ('1' configures the SELECT pulse to precede the first bit of next frame and '0' otherwise).
4. Follow steps 2 to 5 mentioned in **"Enabling and initializing SPI"** on page 164.

19.2.3.3 National Semiconductors SPI

The National Semiconductors' SPI protocol is a half duplex protocol. Rather than transmission and reception occurring at the same time, they take turns. The transmission and reception data sizes may differ. A single "idle" bit transfer period separates transmission from reception. However, the successive data transfers are NOT separated by an idle bit transfer period.

The National Semiconductors SPI protocol only supports mode 0: data is driven on a falling edge of SCLK and data is captured on a rising edge of SCLK.

Figure 19-11 illustrates a single data transfer and two successive data transfers. In both cases the transmission data transfer size is eight bits and the reception data transfer size is four bits.

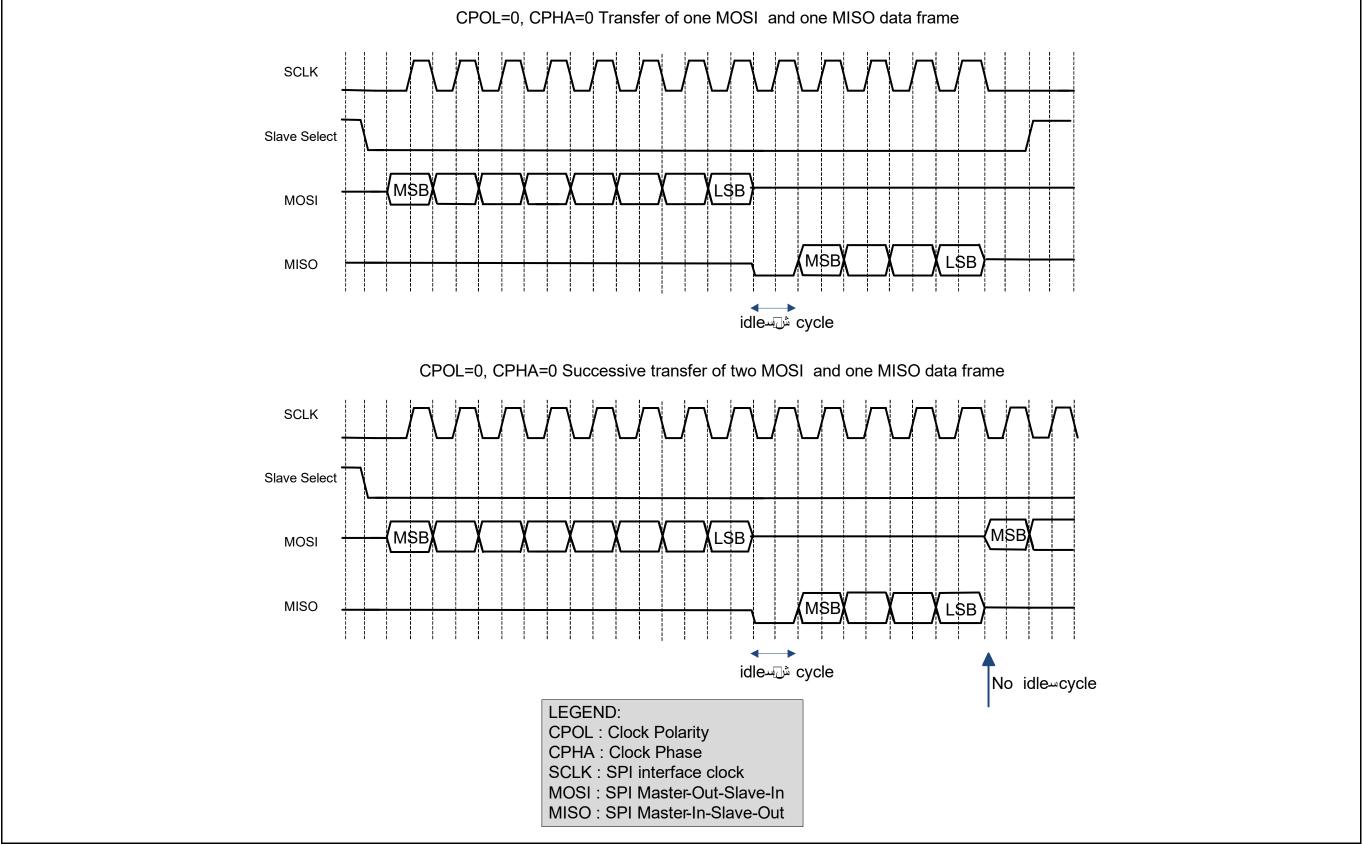

CPOL=0, CPHA=0 Transfer of one MOSI and one MISO data frame
SCLK
Slave Select
MOSI
MSB
LSB
MISO
MSB
LSB
idle cycle
CPOL=0, CPHA=0 Successive transfer of two MOSI and one MISO data frame
SCLK
Slave Select
MOSI
MSB
LSB
MSB
MISO
MSB
LSB
idle cycle
No idle cycle
LEGEND:
CPOL : Clock Polarity
CPHA : Clock Phase
SCLK : SPI interface clock
MOSI : SPI Master-Out-Slave-In
MISO : SPI Master-In-Slave-Out

Figure 19-11. SPI NS data transfer example

19.2.3.3.1 Configuring SCB for SPI NS Mode

To configure the SCB for SPI NS mode, set various register bits in the following order:

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Select SPI NS mode by writing '10' to the MODE (bits [25:24]) of the SCB_SPI_CTRL register.
3. Follow steps 2 to 5 mentioned in **"Enabling and initializing SPI"** on page 164.

19.2.4 Using SPI Master to Clock Slave

In a normal SPI Master mode transmission, the SCLK is generated only when the SCB is enabled and data is being transmitted. This can be changed to always generate a clock on the SCLK line as long as the SCB is enabled. This is used when the slave uses the SCLK for functional operations other than just the SPI functionality. To enable this, write '1' to the SCLK_CONTINUOUS (bit 5) of the SCB_SPI_CTRL register.

19.2.5 Easy SPI protocol

The easy SPI (EZSPI) protocol is based on the Motorola SPI operating in any mode (0, 1, 2, 3). It allows communication between master and slave without the need for CPU intervention at the level of individual frames.

The EZSPI protocol defines an 8-bit EZ address that indexes a memory array (32-entry array of eight bit per entry is supported) located on the slave device. To address these 32 locations, the lower five bits of the EZ address are used. All EZSPI data transfers have 8-bit data frames.

Note: The SCB has a FIFO memory, which is a 16 word by 16-bit SRAM, with byte write enable. The access methods for EZ and non-EZ functions are different. In non-EZ mode, the FIFO is split into TXFIFO and RXFIFO. Each has eight entries of 16 bits per entry. The 16-bit width per entry is used to accommodate configurable data width. In EZ mode, it is used as a single 32x8 bit EZFIFO because only a fixed 8-bit width data is used in EZ mode.

EZSPI has three types of transfers: a write of the EZ address from the master to the slave, a write of data from the master to an addressed slave memory location, and a read by the master from an addressed slave memory location.

19.2.5.1 EZ address write

A write of the EZ address starts with a command byte (0x00) on the MOSI line indicating the master's intent to write the EZ address. The slave then drives a reply byte on the MISO line to indicate that the command is observed (0xFE) or not (0xFF). The second byte on the MOSI line is the EZ address.

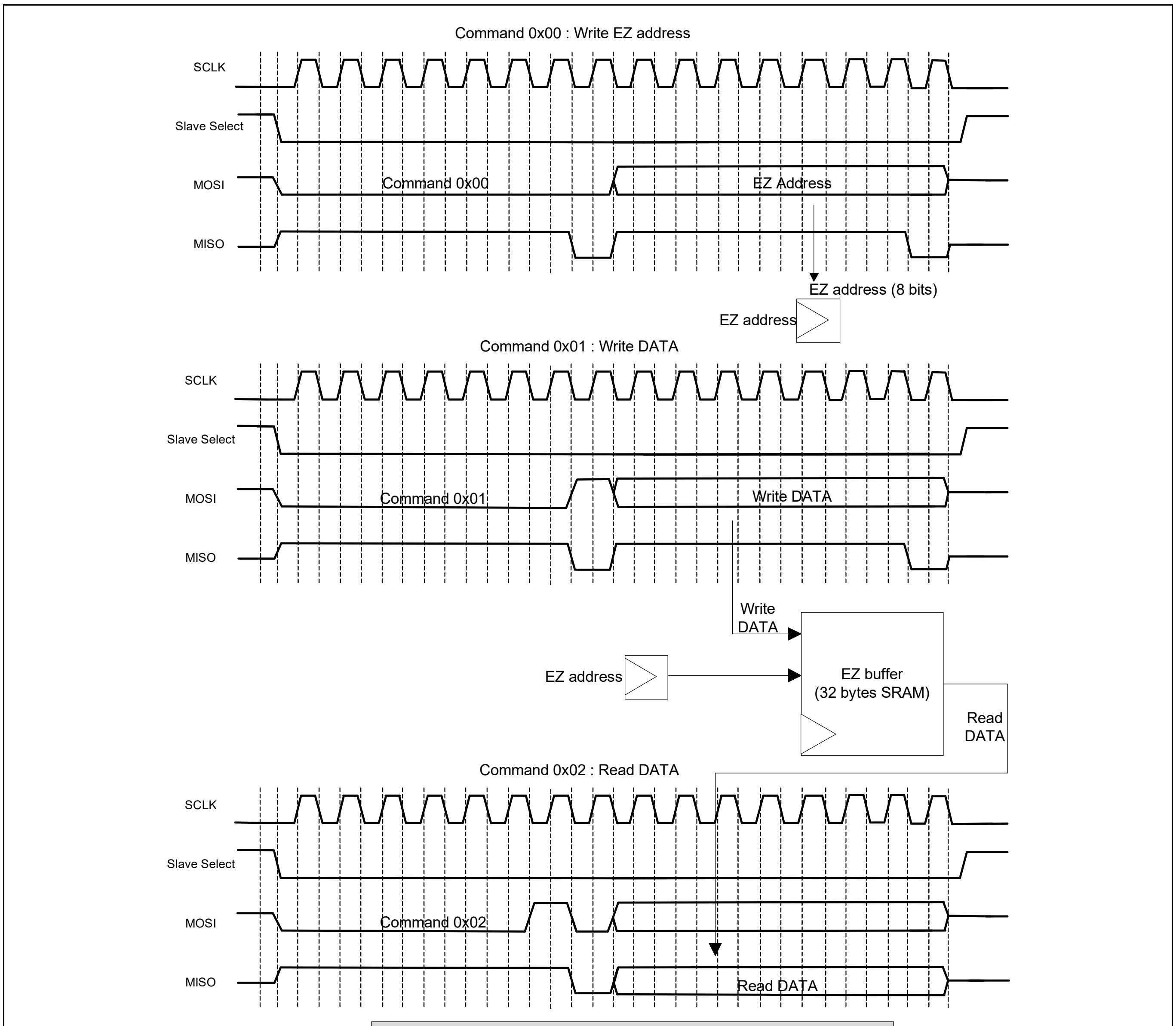
19.2.5.2 Memory array write

A write to a memory array index starts with a command byte (0x01) on the MOSI line indicating the master's intent to write to the memory array. The slave then drives a reply byte on the MISO line to indicate that the command was registered (0xFE) or not (0xFF). Any additional write data bytes on the MOSI line are written to the memory array at locations indicated by the communicated EZ address. The EZ address is automatically incremented by the slave as bytes are written into the memory array. When the EZ address exceeds the maximum number of memory entries (32), it remains there and does not wrap around to 0.

19.2.5.3 Memory array read

A read from a memory array index starts with a command byte (0x02) on the MOSI line indicating the master's intent to read from the memory array. The slave then drives a reply byte on the MISO line to indicate that the command was registered (0xFE) or not (0xFF). Any additional read data bytes on the MISO line are read from the memory array at locations indicated by the communicated EZ address. The EZ address is automatically incremented by the slave as bytes are read from the memory array. When the EZ address exceeds the maximum number of memory entries (32), it remains there and does not wrap around to 0.

Figure 19-12 illustrates the write of EZ address, write to a memory array and read from a memory array operations in the EZSPI protocol.



LEGEND :	
CPOL : Clock Polarity	0x00 : Write EZ address
CPHA : Clock Phase	0x01 : Write DATA
SCLK : SPI Interface Clock	0x02 : Read DATA
MISO : SPI Master-In-Slave-Out	0xFE : slave ready
MOSI : SPI Master-Out-Slave-In	0xFF : slave busy

Figure 19-12. EZSPI example

19.2.6 Configuring SCB for EZSPI Mode

By default, the SCB is configured for non-EZ mode of operation. To configure the SCB for EZSPI mode, set the register bits in the following order:

1. Select EZ mode by writing '1' to the EZ_MODE bit (bit 10) of the SCB_CTRL register.
2. Use continuous transmission mode for the transmitter by writing '1' to the CONTINUOUS bit of SCB_SPI_CTRL register.
3. Follow steps 2 to 5 mentioned in **"Enabling and initializing SPI"** on page 164.

19.2.7 SPI registers

The SPI interface is controlled using a set of 32-bit control and status registers listed in **Table 19-10**. For more information on these registers, see the PSoC™ 4 HV PA *Registers TRM*.

Table 19-10. SPI registers

Register name	Operation
SCB_CTRL	Enables the SCB, selects the type of serial interface (SPI, UART, I^2C), and selects internally and externally clocked operation, EZ and non-EZ modes of operation.
SCB_STATUS	In EZ mode, this register indicates whether the externally clocked logic is potentially using the EZ memory.
SCB_SPI_CTRL	Configures the SPI as either a master or a slave, selects SPI protocols (Motorola, TI, National) and clock-based submodes in Motorola SPI (modes 0,1, 2, 3), selects the type of SELECT signal in TI SPI. When SPI works as slave mode, only the first chip select pin SPI_SELECT[0] can be used in slave mode.
SCB_SPI_STATUS	Indicates whether the SPI bus is busy and sets the SPI slave EZ address in the internally clocked mode.
SCB_TX_CTRL	Specifies the data frame width and specifies whether MSB or LSB is the first bit in transmission.
SCB_RX_CTRL	Performs the same function as that of the SCB_TX_CTRL register, but for the receiver. Also decides whether a median filter is to be used on the input interface lines.
SCB_TX_FIFO_CTRL	Specifies the trigger level, clears the transmitter FIFO and shift registers, and performs the FREEZE operation of the transmitter FIFO.
SCB_RX_FIFO_CTRL	Performs the same function as that of the SCB_TX_FIFO_CTRL register, but for the receiver.
SCB_TX_FIFO_WR	Holds the data frame written into the transmitter FIFO. Behavior is similar to that of a PUSH operation.
SCB_RX_FIFO_RD	Holds the data frame read from the receiver FIFO. Reading a data frame removes the data frame from the FIFO - behavior is similar to that of a POP operation. This register has a side effect when read by software: a data frame is removed from the FIFO.
SCB_RX_FIFO_RD_SILENT	Holds the data frame read from the receiver FIFO. Reading a data frame does not remove the data frame from the FIFO; behavior is similar to that of a PEEK operation.
SCB_RX_MATCH	Holds the slave device address and mask values.

Table 19-10. SPI registers (continued)

Register name	Operation
SCB_TX_FIFO_STATUS	Indicates the number of bytes stored in the transmitter FIFO, the location from which a data frame is read by the hardware (read pointer), the location from which a new data frame is written (write pointer), and decides if the transmitter FIFO holds the valid data.
SCB_RX_FIFO_STATUS	Performs the same function as that of the SCB_TX_FIFO_STATUS register, but for the receiver.
SCB_EZ_DATA	Holds the data in EZ memory location

19.2.8 SPI interrupts

The SPI supports both internal and external interrupt requests. The internal interrupt events are listed here. Custom ISRs can be used by connecting external interrupt component to the interrupt output of the SPI component (with external interrupts enabled).

The SPI predefined interrupts can be classified as TX interrupts and RX interrupts. The TX interrupt output is the logical OR of the group of all possible TX interrupt sources. This signal goes high when any of the enabled TX interrupt sources are true. The RX interrupt output is the logical OR of the group of all possible RX interrupt sources. This signal goes high when any of the enabled Rx interrupt sources are true. Various interrupt registers are used to determine the actual source of the interrupt.

The SPI supports interrupts on the following events:

- SPI master transfer done
- SPI Bus Error - Slave deselected at an unexpected time in the SPI transfer
- SPI slave deselected after any EZSPI transfer occurred
- SPI slave deselected after a write EZSPI transfer occurred
- TX
  - TX FIFO has less entries than the value specified by TRIGGER_LEVEL in SCB_TX_FIFO_CTRL
  - TX FIFO is not full
  - TX FIFO is empty
  - TX FIFO overflow
  - TX FIFO underflow
- RX
  - RX FIFO is full
  - RX FIFO is not empty
  - RX FIFO overflow
  - RX FIFO underflow
- SPI Externally clocked
  - Wake up request on slave select
  - SPI STOP detection at the end of each transfer
  - SPI STOP detection at the end of a write transfer
  - SPI STOP detection at the end of a read transfer

Note The SPI interrupt signal is hard-wired to the Cortex-M0 NVIC and cannot be routed to external pins.

19.2.9 Enabling and initializing SPI

The SPI must be programmed in the following order:

1. Program protocol specific information using the SCB_SPI_CTRL register, according to **Table 19-15**. This includes selecting the submodes of the protocol and selecting master-slave functionality. EZSPI can be used with slave mode only.
2. Program the generic transmitter and receiver information using the SCB_TX_CTRL and SCB_RX_CTRL registers, as shown in **Table 19-16**:
   a) Specify the data frame width. This should always be 8 for EZSPI.
   b) Specify whether MSB or LSB is the first bit to be transmitted/received. This should always be MSB first for EZSPI.
3. Program the transmitter and receiver FIFOs using the SCB_TX_FIFO_CTRL and SCB_RX_FIFO_CTRL registers respectively, as shown in **Table 19-17**:
   a) Set the trigger level.
   b) Clear the transmitter and receiver FIFO and Shift registers.
   c) Freeze the TX and RX FIFO.
4. Program SCB_CTRL register to enable the SCB block. Also select the mode of operation. These register bits are shown in **Table 19-11**.
5. Enable the block (write a '1' to the ENABLED bit of the SCB_CTRL register). After the block is enabled, control bits should not be changed. Changes should be made after disabling the block; for example, to modify the operation mode (from Motorola mode to TI mode) or to go from externally clocked to internally clocked operation. The change takes effect only after the block is re-enabled. Note that re-enabling the block causes re-initialization and the associated state is lost (for example, FIFO content).

Table 19-11. SCB_CTRL Register

Bits	Name	Value	Description
[25:24]	MODE	00	I^{2}C mode
		01	SPI mode
		10	UART mode
		11	Reserved
31	ENABLED	0	SCB block disabled
		1	SCB block enabled

Table 19-12. SCB_SPI_CTRL Register

Bits	Name	Value	Description
[25:24]	MODE	00	SPI Motorola submode. (This is the only mode supported for EZSPI.)
		01	SPI Texas Instruments submode.
		10	SPI National Semiconductors submode.
		11	Reserved.
31	MASTER_MODE	0	Slave mode. (This is the only mode supported for EZSPI.)
		1	Master mode.

Table 19-13. SCB_TX_CTRL/SCB_RX_CTRL Registers

Bits	Name	Description
[3:0]	DATA_ WIDTH	'DATA_WIDTH + 1' is the number of bits in the transmitted or received data frame. The valid range is [3, 15]. This does not include start, stop, and parity bits. For EZSPI, this value should be '0b0111'.
8	MSB_FIRST	1 = MSB first 0 = LSB first For EZSPI, this value should be 1.
9	MEDIAN	This is for SCB_RX_CTRL only. Decides whether a digital three-tap median filter is applied on the input interface lines. This filter should reduce susceptibility to errors, but it requires higher oversampling values. 1 = Enabled 0 = Disabled

Table 19-14. SCB_TX_FIFO_CTRL/SCB_RX_FIFO_CTRL Registers

Bits	Name	Description
[3:0]	TRIGGER_LEVEL	Trigger level. When the transmitter FIFO has less entries or receiver FIFO has more entries than the value of this field, a transmitter or receiver trigger event is generated in the respective case.
16	CLEAR	When '1', the transmitter or receiver FIFO and the shift registers are cleared.
17	FREEZE	When '1', hardware reads/writes to the transmitter or receiver FIFO have no effect. Freeze does not advance the TX or RX FIFO read/write pointer.

19.2.10 Internally and externally clocked SPI operations

The SCB supports both internally and externally clocked operations for SPI and I^2C functions. An internally clocked operation uses a clock provided by the chip. An externally clocked operation uses a clock provided by the serial interface. Externally clocked operation enables operation in the Deep Sleep system power mode.

Internally clocked operation uses the high-frequency clock (HFCLK) of the system. For more information on system clocking, see the **"Clocking system"** on page 93. It also supports oversampling. Oversampling is implemented with respect to the high-frequency clock. The OVS (bits [3:0]) of the SCB_CTRL register specify the oversampling.

In SPI master mode, the valid range for oversampling is 4 to 16. Hence, with a clock speed of 48 MHz, the maximum bit rate is 12 Mbps. However, if you consider the I/O cell and routing delays, the oversampling must be set between 6 and 16 for proper operation. So, the maximum bit rate is 8 Mbps.

Note To achieve maximum possible bit rate, LATE_MISO_SAMPLE must be set to '1' in SPI master mode. This has a default value of '0'.

In SPI slave mode, the OVS field (bits [3:0]) of SCB_CTRL register is not used. However, there is a frequency requirement for the SCB clock with respect to the interface clock (SCLK). This requirement is expressed in terms of the ratio (SCB clock/SCLK). This ratio is dependent on two fields: MEDIAN of SCB_RX_CTRL register and LATE_MISO_SAMPLE of SCB_CTRL register. If the external SPI master supports Late MISO sampling and if the median bit is set to '0', the maximum data rate that can be achieved is 8 Mbps. If the external SPI master does not support late MISO sampling, the maximum data rate is limited to 4 Mbps (with the median bit set to '0'). Based on these bits, the maximum bit rates are given in **Table 19-15**.

Table 19-15. SPI Slave Maximum Data Rates

Maximum Bit Rate at Peripheral Clock of 48 MHz	Ratio requirement	Median of SCB_RX_CTRL	LATE_MISO_SAMPLE of SCB_CTRL
8 Mbps	≥6	0	1
6 Mbps	≥8	1	1
4 Mbps	≥12	0	0
3 Mbps	≥16	1	0

Externally clocked operation is limited to:

- Slave functionality.
- EZ functionality. EZ functionality uses the block's SRAM as a memory structure. Non-EZ functionality uses the block's SRAM as TX and RX FIFOs; FIFO support is not available in externally clocked operation.
- Motorola mode 0, 1, 2, 3.

Externally clocked EZ mode of operation can support a data rate of 48 Mbps (at the interface clock of 48 MHz).

Internally and externally clocked operation is determined by two register fields of the SCB_CTRL register:

- **EC_AM_MODE:** Indicates whether SPI slave selection is internally ('0') or externally ('1') clocked. SPI slave selection comprises the first part of the protocol.
- **EC_OP_MODE:** Indicates whether the rest of the protocol operation (besides SPI slave selection) is internally ('0') or externally ('1') clocked. As mentioned earlier, externally clocked operation does NOT support non-EZ functionality.

These two register fields determine the functional behavior of SPI. The register fields should be set based on the required behavior in Active, Sleep, and Deep Sleep system power mode. Improper setting may result in faulty behavior in certain system power modes. **Table 19-16** and **Table 19-17** describe the settings for SPI (in non-EZ and EZ modes).

19.2.10.1 Non-EZ mode of operation

In non-EZ mode there are two possible settings. As externally clocked operation is not supported for non-EZ functionality (no FIFO support), EC_OP_MODE should always be set to ‘0’. However, EC_AM_MODE can be set to ‘0’ or ‘1’. **Table 19-16** gives an overview of the possibilities.

Table 19-16. SPI Operation in Non-EZ Mode

SPI (non-EZ) Mode				
System Power Mode	**EC_OP_MODE = 0**		**EC_OP_MODE = 1**	
	EC_AM_MODE = 0	**EC_AM_MODE = 1**	**EC_AM_MODE = 0**	**EC_AM_MODE = 1**
Active and Sleep	Selection using internal clock. Operation using internal clock.	Selection using external clock: Operation using internal clock. In Active mode, the Wakeup interrupt cause is disabled (MASK = 0). In Sleep mode, the MASK bit can be configured by the user.	Not supported	Not supported
Deep Sleep	Not supported	Selection using external clock: Wakeup interrupt cause is enabled (MASK = 1). Send 0xFF.		

EC_OP_MODE is ‘0’ and EC_AM_MODE is ‘0’: This setting only works in Active and Sleep system power modes. The entire block's functionality is provided in the internally clocked domain.

EC_OP_MODE is ‘0’ and EC_AM_MODE is ‘1’: This setting works in Active and Sleep system power mode and provides limited (wake up) functionality in Deep Sleep system power mode. SPI slave selection is performed by the externally clocked logic: in Active system power mode, both internally and externally clocked logic are active, and in Deep Sleep system power mode, only the externally clocked logic is active. When the externally clocked logic detects slave selection, it sets a wakeup interrupt cause bit, which can be used to generate an interrupt to wake up the CPU.

- In Active system power mode, the CPU and the block’s internally clocked operation are active and the wakeup interrupt cause is disabled (associated MASK bit is ‘0’). But in the Sleep mode, wakeup interrupt cause can be either enabled or disabled (MASK bit can be either ‘1’ or ‘0’) based on the application. The remaining operations in the Sleep mode are same as that of the Active mode. The internally clocked operation takes care of the ongoing SPI transfer.
- In Deep Sleep system power mode, the CPU needs to be woken up and the wakeup interrupt cause is enabled (MASK bit is ‘1’). Waking up takes time, so the ongoing SPI transfer is negatively acknowledged (‘1’ bit or 0xFF byte is sent out on the MISO line) and the internally clocked operation takes care of the next SPI transfer when it is woken up.

19.2.10.2 EZ mode of operation

EZ mode has three possible settings. EC_AM_MODE can be set to ‘0’ or ‘1’ when EC_OP_MODE is ‘0’ and EC_AM_MODE must be set to ‘1’ when EC_OP_MODE is ‘1’. **Table 19-17** gives an overview of the possibilities. The gray cells indicate a possible, yet not recommended, setting because it involves a switch from the externally clocked logic (slave selection) to the internally clocked logic (rest of the operation). The combination EC_AM_MODE = 0 and EC_OP_MODE = 1 is invalid and the block will not respond.

Table 19-17. SPI Operation in EZ Mode

SPI, EZ Mode				
System Power Mode	**EC_OP_MODE = 0**		**EC_OP_MODE = 1**	
	EC_AM_MODE = 0	**EC_AM_MODE = 1**	**EC_AM_MODE = 0**	**EC_AM_MODE = 1**
Active and Sleep	Selection using internal clock. Operation using internal clock.	Selection using external clock. Operation using internal clock. In Active mode, the Wakeup interrupt cause is disabled (MASK = 0). In Sleep mode, the MASK bit can be configured by the user.	Invalid	Selection using external clock. Operation using external clock.
Deep Sleep	Not supported	Selection using external clock: Wakeup interrupt cause is enabled (MASK = 1). Send 0xFF.		Selection using external clock. Operation using external clock.

EC_OP_MODE is '0' and EC_AM_MODE is '0': This setting only works in Active and Sleep system power modes. The entire block's functionality is provided in the internally clocked domain.

EC_OP_MODE is '0' and EC_AM_MODE is '1': This setting works in Active and Sleep system power modes and provides limited (wake up) functionality in Deep Sleep system power mode. SPI slave selection is performed by the externally clocked logic: in Active system power mode, both internally and externally clocked logic are active, and in Deep Sleep system power mode, only the externally clocked logic is active. When the externally clocked logic detects slave selection, it sets a wakeup interrupt cause bit, which can be used to generate an interrupt to wake up the CPU.

- In Active system power mode, the CPU and the block's internally clocked operation are active and the wakeup interrupt cause is disabled (associated MASK bit is '0'). But in Sleep mode, wakeup interrupt cause can be either enabled or disabled (MASK bit can be either '1' or '0') based on the application. The remaining operations in the Sleep mode are same as that of the Active mode. The internally clocked operation takes care of the ongoing SPI transfer.
- In Deep Sleep system power mode, the CPU needs to be woken up and the wakeup interrupt cause is enabled (MASK bit is '1'). Waking up takes time, so the ongoing SPI transfer is negatively acknowledged ('1' bit or 0xFF byte is sent out on the MISO line) and the internally clocked operation takes care of the next SPI transfer when it is woken up.

EC_OP_MODE is '1' and EC_AM_MODE is '1': This setting works in Active, Sleep, and Deep Sleep system power modes. The SCB functionality is provided in the externally clocked domain. Note that this setting results in externally clocked accesses to the block's SRAM. These accesses may conflict with internally clocked accesses from the device. This may cause wait states or bus errors. The field FIFO_BLOCK of the SCB_CTRL register determines whether wait states ('1') or bus errors ('0') are generated.

19.3 UART

The Universal Asynchronous Receiver/Transmitter (UART) protocol is an asynchronous serial interface protocol. UART communication is typically point-to-point. The UART interface consists of two signals:

- TX: Transmitter output
- RX: Receiver input

19.3.1 Features

- Asynchronous transmitter and receiver functionality
- Supports a maximum data rate of 3 Mbps
- Supports UART protocol
  - Standard UART
  - SmartCard (ISO7816) reader.
  - IrDA
- Supports Local Interconnect Network (LIN)
  - Break detection
  - Baud rate detection
  - Collision detection (ability to detect that a driven bit value is not reflected on the bus, indicating that another component is driving the same bus)
- Multi-processor mode
- Data frame size programmable from 4 to 9 bits
- Programmable number of STOP bits, which can be set in terms of half bit periods between 1 and 4
- Parity support (odd and even parity)
- Interrupt or polling CPU interface
- Programmable oversampling

19.3.2 General description

Figure 19-13 illustrates a standard UART TX and RX.

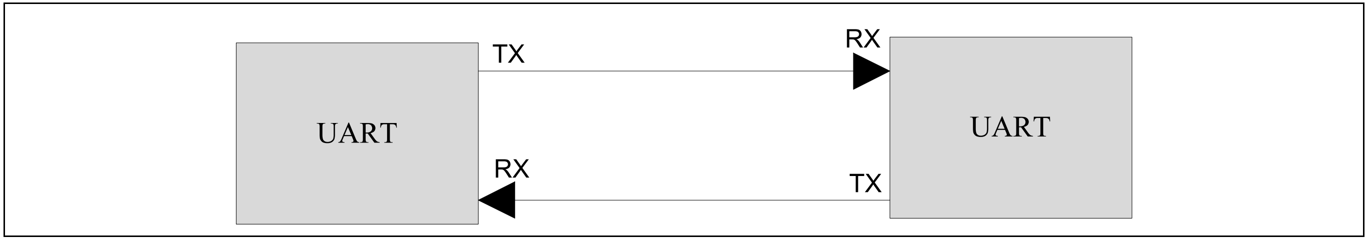


Figure 19-13. UART example

A typical UART transfer consists of a “Start Bit” followed by multiple “Data Bits”, optionally followed by a “Parity Bit” and finally completed by one or more “Stop Bits”. The Start and Stop bits indicate the start and end of data transmission. The Parity bit is sent by the transmitter and is used by the receiver to detect single bit errors. As the interface does not have a clock (asynchronous), the transmitter and receiver use their own clocks; also, they need to agree upon the period of a bit transfer.

Three different serial interface protocols are supported:

- Standard UART protocol
  - Multi-Processor Mode
  - Local Interconnect Network (LIN)
- SmartCard, similar to UART, but with a possibility to send a negative acknowledgement
- IrDA, modification to the UART with a modulation scheme

By default, UART supports a data frame width of eight bits. However, this can be configured to any value in the range of 4 to 9. This does not include start, stop, and parity bits. The number of stop bits can be in the range of 1 to 4. The parity bit can be either enabled or disabled. If enabled, the type of parity can be set to either even parity or odd parity. The option of using the parity bit is available only in the Standard UART and SmartCard UART modes. For IrDA UART mode, the parity bit is automatically disabled. Figure 19-18 depicts the default configuration of the UART interface of the SCB.

Note: UART interface does not support external clocking operation. Hence, UART operates only in the Active and Sleep system power modes.

19.3.3 UART modes of operation

19.3.3.1 Standard protocol

A typical UART transfer consists of a start bit followed by multiple data bits, optionally followed by a parity bit and finally completed by one or more stop bits. The start bit value is always '0', the data bits values are dependent on the data transferred, the parity bit value is set to a value guaranteeing an even or odd parity over the data bits, and the stop bit value is '1'. The parity bit is generated by the transmitter and can be used by the receiver to detect single bit transmission errors. When not transmitting data, the TX line is '1' – the same value as the stop bits.

Because the interface does not have a clock, the transmitter and receiver need to agree upon the period of a bit transfer. The transmitter and receiver have their own internal clocks. The receiver clock runs at a higher frequency than the bit transfer frequency, such that the receiver may oversample the incoming signal.

The transition of a stop bit to a start bit is represented by a change from '1' to '0' on the TX line. This transition can be used by the receiver to synchronize with the transmitter clock. Synchronization at the start of each data transfer allows error-free transmission even in the presence of frequency drift between transmitter and receiver clocks. The required clock accuracy is dependent on the data transfer size.

The stop period or the amount of stop bits between successive data transfers is typically agreed upon between transmitter and receiver, and is typically in the range of 1 to 3-bit transfer periods.

Figure 19-14 illustrates the UART protocol.

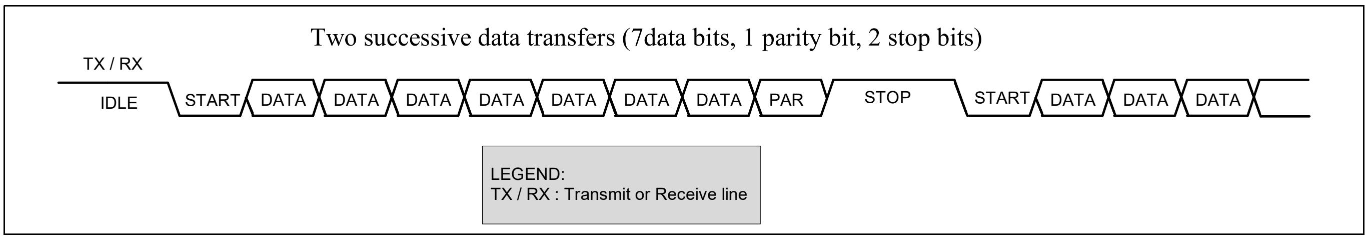

Two successive data transfers (7data bits, 1 parity bit, 2 stop bits)
TX / RX
IDLE
START
DATA
DATA
DATA
DATA
DATA
DATA
DATA
PAR
STOP
START
DATA
DATA
DATA
LEGEND:
TX / RX : Transmit or Receive line

Figure 19-14. UART, Standard Protocol example

The receiver oversamples the incoming signal; the value of the sample point in the middle of the bit transfer period (on the receiver's clock) is used. **Figure 19-15** illustrates this.

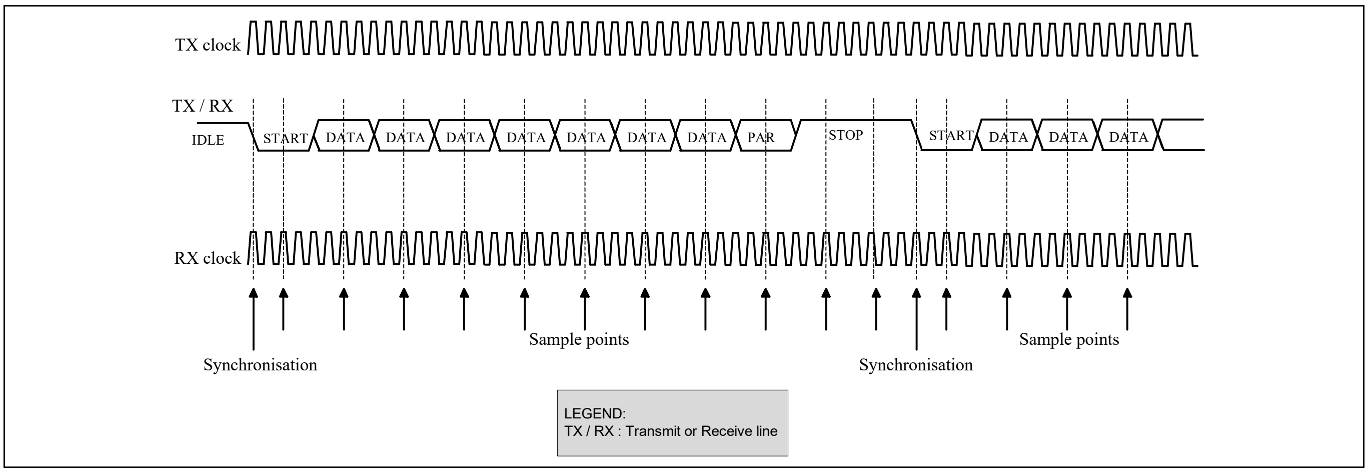


Figure 19-15. UART, Standard Protocol example (Single Sample)

Alternatively, three samples around the middle of the bit transfer period (on the receiver's clock) are used for a majority vote to increase accuracy. **Figure 19-16** illustrates this.

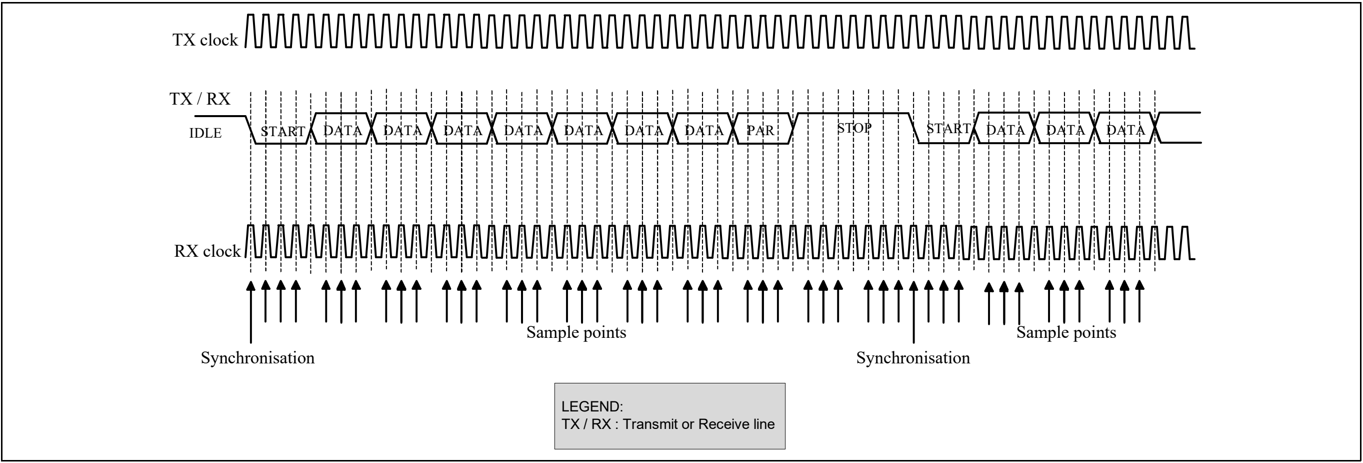


Figure 19-16. UART, Standard Protocol (Multiple Samples)

19.3.3.1.1 UART Multi-Processor Mode

The UART_MP (multi-processor) mode is defined with single-master-multi-slave topology, as **Figure 19-17** shows. This mode is also known as UART 9-bit protocol because the data field is nine bits wide. UART_MP is part of Standard UART mode.

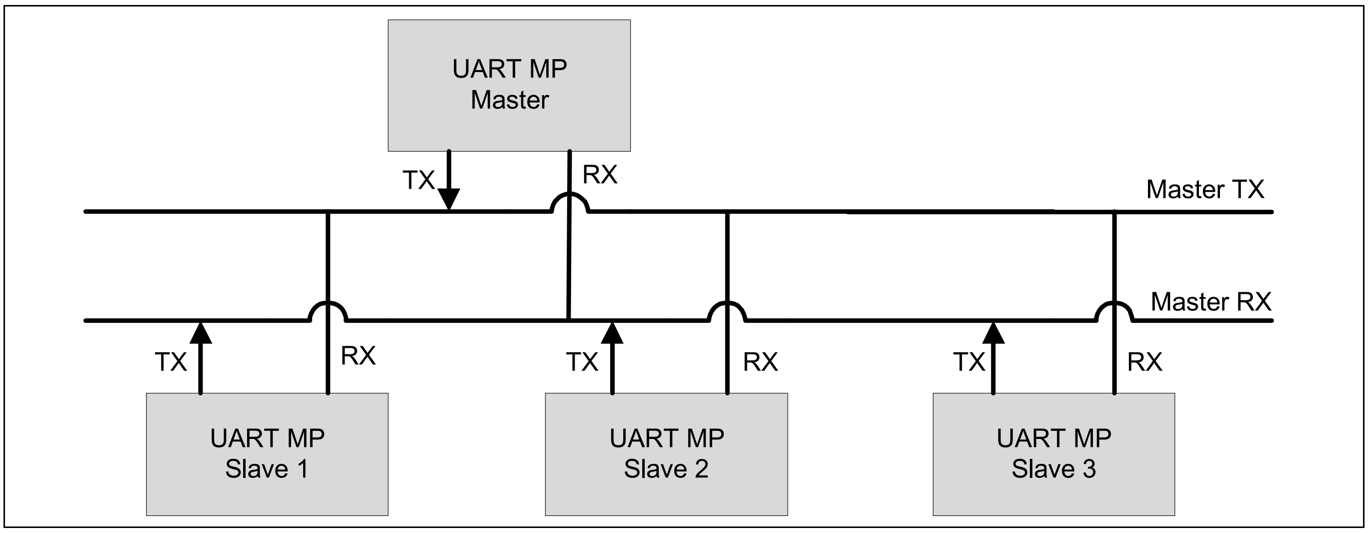


Figure 19-17. UART MP mode bus connections

The main properties of UART_MP mode are:

- Single master with multiple slave concept (multi-drop network).
- Each slave is identified by a unique address.
- Using 9-bit data field, with the ninth bit as address/data flag (MP bit). When set high, it indicates an address byte; when set low it indicates a data byte. A data frame is illustrated in **Figure 19-18**.
- Parity bit is disabled.

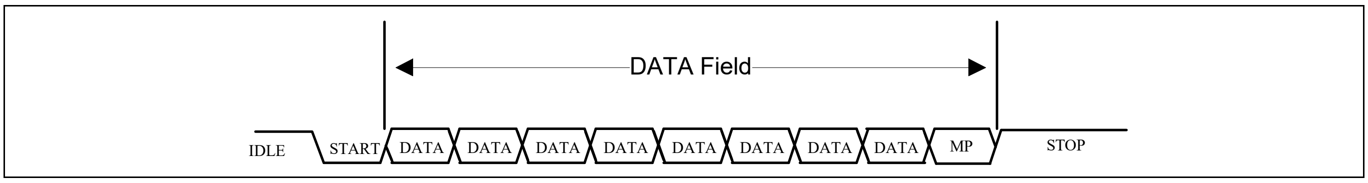


Figure 19-18. UART MP data frame

The SCB can be used as either master or slave device in UART_MP mode. Both SCB_TX_CTRL and SCB_RX_CTRL registers should be set to 9-bit data frame size. When the SCB works as UART_MP master device, the firmware changes the MP flag for every address or data frame. When it works as UART_MP slave device, the MP_MODE field of the SCB_UART_RX_CTRL register should be set to ‘1’. The SCB_RX_MATCH register should be set for the slave address and address mask. The matched address is written in the RX_FIFO when ADDR_ACCEPT field of the SCB_CTRL register is set to ‘1’. If received address does not match its own address, then the interface ignores the following data, until next address is received for compare.

19.3.3.1.2 UART Local Interconnect Network (LIN) Mode

The LIN protocol is supported by the SCB as part of the standard UART. LIN is designed with single-master-multi-slave topology. There is one master node and multiple slave nodes on the LIN bus. The SCB UART supports both LIN master and slave functionality. The LIN specification defines both physical layer (layer 1) and data link layer (layer 2). **Figure 19-19** illustrates the UART_LIN and LIN Transceiver.

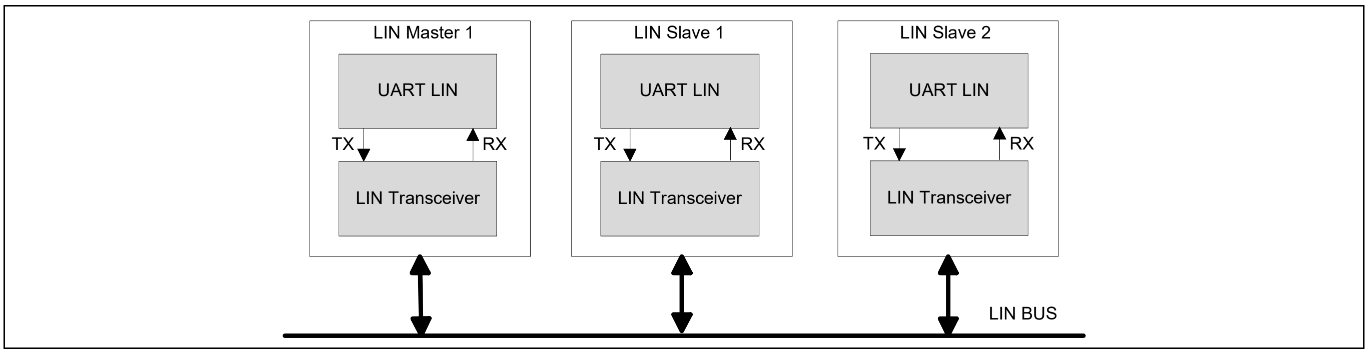


Figure 19-19. UART_LIN and LIN Transceiver

LIN protocol defines two tasks:

- Master task: This task involves sending a header packet to initiate a LIN transfer.
- Slave task: This task involves transmitting or receiving a response.

The master node supports master task and slave task; the slave node supports only slave task, as shown in **Figure 19-20**.

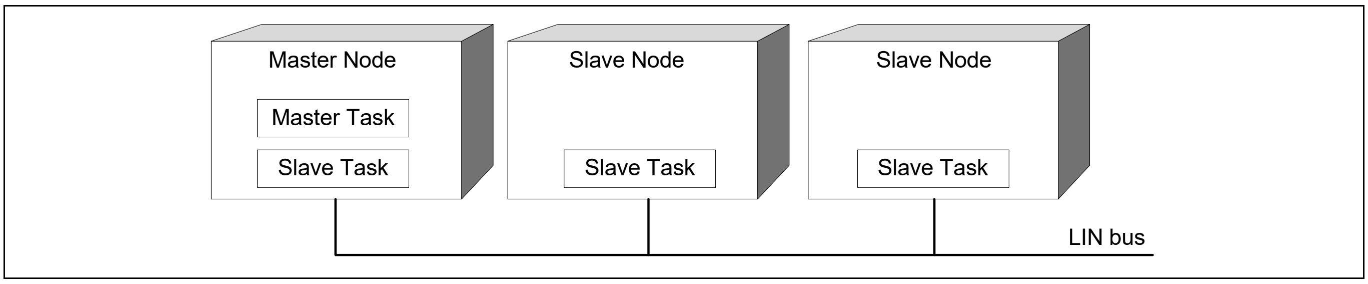


Figure 19-20. LIN Bus Nodes and Tasks

19.3.3.1.3 LIN Frame Structure

LIN is based on the transmission of frames at predetermined moments of time. A frame is divided into header and response fields, as shown in **Figure 19-21**.

- The header field consists of:
  - Break field (at least 13 bit periods with the value '0').
  - Sync field (a 0x55 byte frame). A sync field can be used to synchronize the clock of the slave task with that of the master task.
  - Identifier field (a frame specifying a specific slave).
- The response field consists of data and checksum.

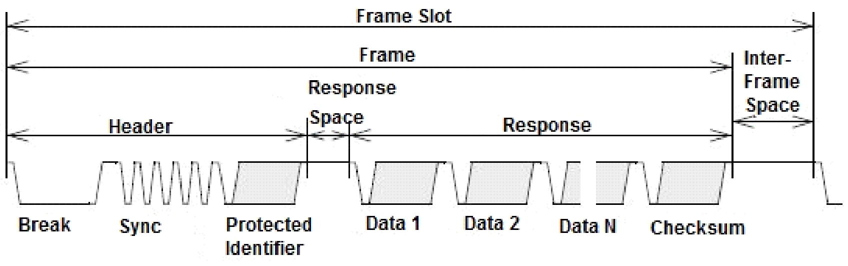


Figure 19-21. LIN Frame Structure

In LIN protocol communication, the least significant bit (LSB) of the data is sent first and the most significant bit (MSB) last. The start bit is encoded as zero and the stop bit is encoded as one. The following sections describe all the byte fields in the LIN frame.

Break Field

Every new frame starts with a break field, which is always generated by the master. The break filed has logical zero with a minimum of 13 bit times and followed by a break delimiter. The break field structure is as shown in **Figure 19-22**.

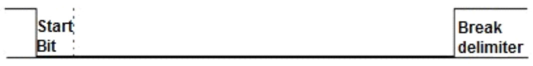


Figure 19-22. LIN Break Field

Sync Field

This is the second field transmitted by the master in the header field; its value is 0x55. A sync field can be used to synchronize the clock of the slave task with that of the master task for automatic baud rate detection. **Figure 19-23** shows the LIN sync field structure.

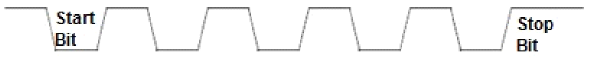


Figure 19-23. LIN Sync Field

Protected identifier (PID) Field

A protected identifier field consists of two sub-fields: the frame identifier (bits 0–5) and the parity (bit 6 and bit 7).

- Frame identifier: The frame identifiers are split into three categories
  - Values 0 to 59 (0x3B) are used for signal carrying frames
  - 60 (0x3C) and 61 (0x3D) are used to carry diagnostic and configuration data
  - 62 (0x3E) and 63 (0x3F) are reserved for future protocol enhancements
- Parity: Frame identifier bits are used to calculate the parity

Figure 19-24 shows the PID field structure.

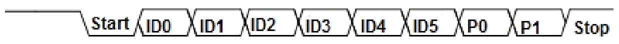


Figure 19-24. PID Field

Data

In LIN, every frame can carry a minimum of one byte and maximum of 8 bytes of data. Here, the LSB of the data byte is sent first and the MSB of the data byte is sent last.

Checksum

The checksum is the last byte field in the LIN frame. It is calculated by inverting the 8-bit sum along with carryover of all data bytes only or the 8-bit sum with the carryover of all data bytes and the PID field. The two types of checksums in LIN frames are:

- Classic checksum: the checksum calculated over all the data bytes only (used in LIN 1.x slaves).
- Enhanced checksum: the checksum calculated over all the data bytes along with the protected identifier (used in LIN 2.x slaves).

LIN Frame Types

The type of frame refers to the conditions that need to be valid to transmit the frame. According to the LIN specification, there are five different types of LIN frames. A node or cluster does not have to support all frame types.

Unconditional Frame

These frames carry the signals and their frame identifiers (of 0x00 to 0x3B range). The subscriber will receive the frames and make it available to the application; the publisher of the frame will provide the response to the header.

Event-Triggered Frame

The purpose of an event-triggered frame is to increase the responsiveness of the LIN cluster without assigning too much of the bus bandwidth to polling of multiple slave nodes with seldom occurring events. Event-triggered frames carry the response of one or more unconditional frames. The unconditional frames associated with an event triggered frame should:

- Have equal length
- Use the same checksum model (either classic or enhanced)
- Reserve the first data field to its protected identifier
- Be published by different slave nodes
- Not be included directly in the same schedule table as the event-triggered frame

Sporadic Frame

The purpose of the sporadic frames is to merge some dynamic behavior into the schedule table without affecting the rest of the schedule table. These frames have a group of unconditional frames that share the frame slot. When the sporadic frame is due for transmission, the unconditional frames are checked if they have any updated signals. If no signals are updated, no frame will be transmitted and the frame slot will be empty.

Diagnostic Frames

Diagnostic frames always carry transport layer, and contains eight data bytes.

The frame identifier for diagnostic frame is:

- Master request frame (0x3C), or
- Slave response frame (0x3D)

Before transmitting a master request frame, the master task queries its diagnostic module to see if it will be transmitted or if the bus will be silent. A slave response frame header will be sent unconditionally. The slave tasks publish and subscribe to the response according to their diagnostic modules.

Reserved Frames

These frames are reserved for future use; their frame identifiers are 0x3E and 0x3F.

LIN Go-To-Sleep and Wake-Up

The LIN protocol has the feature of keeping the LIN bus in Sleep mode, if the master sends the go-to-sleep command. The go-to-sleep command is a master request frame (ID = 0x3C) with the first byte field is equal to 0x00 and rest set to 0xFF. The slave node application may still be active after the go-to-sleep command is received. This behavior is application specific. The LIN slave nodes automatically enter Sleep mode if the LIN bus inactivity is more than four seconds.

Wake-up can be initiated by any node connected to the LIN bus – either LIN master or any of the LIN slaves by forcing the bus to be dominant for 250 µs to 5 ms. Each slave should detect the wakeup request and be ready to process headers within 100 ms. The master should also detect the wakeup request and start sending headers when the slave nodes are active.

To support LIN, a dedicated (off-chip) line driver/receiver is required. Supply voltage range on the LIN bus is 7 V to 18 V. Typically, LIN line drivers will drive the LIN line with the value provided on the SCB TX line and present the value on the LIN line to the SCB RX line. By comparing TX and RX lines in the SCB, bus collisions can be detected (indicated by the SCB_UART_ARB_LOST field of the SCB_INTR_TX register).

Configuring the SCB as Standard UART Interface

To configure the SCB as a standard UART interface, set various register bits in the following order:

1. Configure the SCB as UART interface by writing ‘10’ to the MODE field (bits [25:24]) of the SCB_CTRL register.
2. Configure the UART interface to operate as a Standard protocol by writing ‘00’ to the MODE field (bits [25:24]) of the SCB_UART_CTRL register.
3. To enable the UART MP Mode or UART LIN Mode, write ‘1’ to the MP_MODE (bit 10) or LIN_MODE (bit 12) respectively of the SCB_UART_RX_CTRL register.
4. Follow steps 2 to 5 described in **“Enabling and initializing UART”** on page 183.

19.3.3.2 SmartCard (ISO7816)

ISO7816 is asynchronous serial interface, defined with single-master-single slave topology. ISO7816 defines both Reader (master) and Card (slave) functionality. For more information, refer to the ISO7816 Specification. Only master (reader) function is supported by the SCB. This block provides the basic physical layer support with asynchronous character transmission. UART_TX line is connected to SmartCard I/O line, by internally multiplexing between UART_TX and UART_RX control modules.

The SmartCard transfer is similar to a UART transfer, with the addition of a negative acknowledgement (NACK) that may be sent from the receiver to the transmitter. A NACK is always ‘0’. Both master and slave may drive the same line, although never at the same time.

A SmartCard transfer has the transmitter drive the start bit and data bits (and optionally a parity bit). After these bits, it enters its stop period by releasing the bus. Releasing results in the line being ‘1’ (the value of a stop bit). After one bit transfer period into the stop period, the receiver may drive a NACK on the line (a value of ‘0’) for one bit transfer period. This NACK is observed by the transmitter, which reacts by extending its stop period by one bit transfer period. For this protocol to work, the stop period should be longer than one bit transfer period. Note that a data transfer with a NACK takes one bit transfer period longer, than a data transfer without a NACK. Typically, implementations use a tristate driver with a pull-up resistor, such that when the line is not transmitting data or transmitting the Stop bit, its value is ‘1’.

Figure 19-25 illustrates the SmartCard protocol.

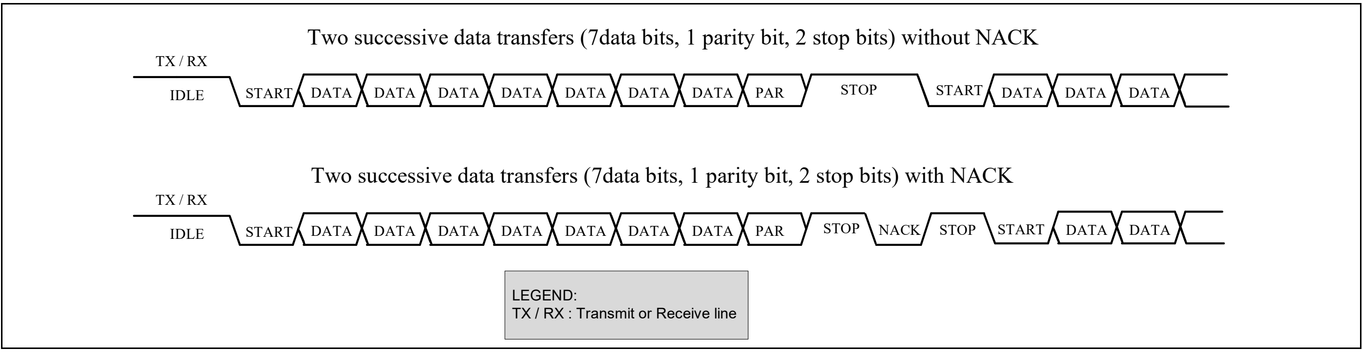

Two successive data transfers (7data bits, 1 parity bit, 2 stop bits) without NACK
TX / RX
IDLE
START
DATA
PAR
STOP
Two successive data transfers (7data bits, 1 parity bit, 2 stop bits) with NACK
NACK
LEGEND:
TX / RX : Transmit or Receive line

Figure 19-25. SmartCard example

The communication Baud rate for ISO7816 is given as:

Baud rate= $f_{7816} \times (D/F)$

Where f_{7816} is the clock frequency, F is the clock rate conversion integer, and D is the baud rate adjustment integer.

By default, F = 372, D = f1, and the maximum clock frequency is 5 MHz. Thus, maximum baud rate is 13.4 Kbps. Typically, a 3.57-MHz clock is selected. The typical value of the baud rate is 9.6 Kbps.

19.3.3.2.1 Configuring SCB as UART SmartCard Interface

To configure the SCB as a UART SmartCard interface, set various register bits in the following order.

1. Configure the SCB as UART interface by writing ‘10’ to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Configure the UART interface to operate as a SmartCard protocol by writing ‘01’ to the MODE (bits [25:24]) of the SCB_UART_CTRL register.
3. Follow steps 2 to 5 described in **“Enabling and initializing UART”** on page 183.

19.3.3.3 IrDA

The SCB supports the Infrared Data Association (IrDA) protocol for data rates of up to 115.2 Kbps using the UART interface. It supports only the basic physical layer of IrDA protocol with rates less than 115.2 Kbps. Hence, the system instantiating this block must consider how to implement a complete IrDA communication system with other available system resources.

The IrDA protocol adds a modulation scheme to the UART signaling. At the transmitter, bits are modulated. At the receiver, bits are demodulated. The modulation scheme uses a Return-to-Zero-Inverted (RZI) format. A bit value of ‘0’ is signaled by a short ‘1’ pulse on the line and a bit value of ‘1’ is signaled by holding the line to ‘0’. For these data rates (<= 115.2 Kbps), the RZI modulation scheme is used and the pulse duration is 3/16 of the bit period. The sampling clock frequency should be set 16 times the selected baud rate, by configuring the SCB_OVS field of the SCB_CTRL register.

Different communication speeds under 115.2 Kbps can be achieved by configuring corresponding block clock frequency. Additional allowable rates are 2.4 Kbps, 9.6 Kbps, 19.2 Kbps, 38.4 Kbps, and 57.6 Kbps. An IrDA serial infrared interface operates at 9.6 Kbps. **Figure 19-26** shows how a UART transfer is IrDA modulated.

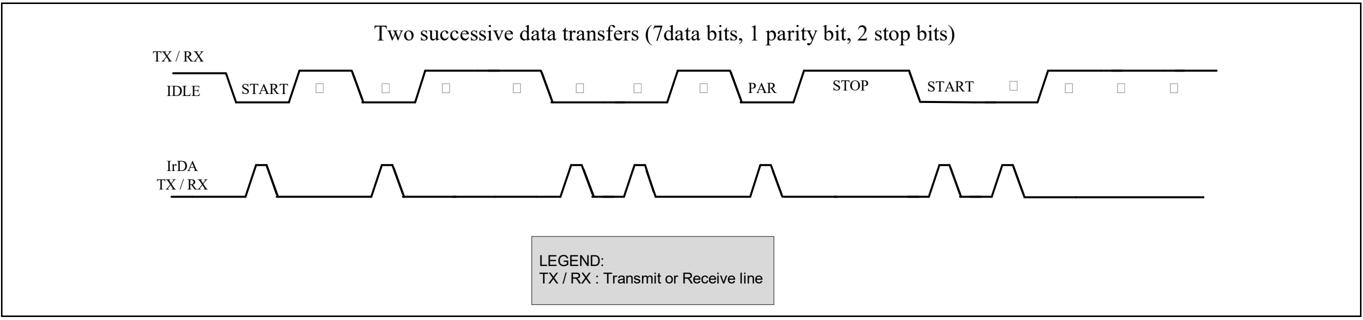

Two successive data transfers (7data bits, 1 parity bit, 2 stop bits)
TX / RX
IDLE
START
PAR
STOP
START
IrDA
TX / RX
LEGEND:
TX / RX : Transmit or Receive line

Figure 19-26. IrDA example

19.3.3.3.1 Configuring the SCB as UART IrDA Interface

To configure the SCB as a UART IrDA interface, set various register bits in the following order.

1. Configure the SCB as UART interface by writing ‘10’ to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Configure the UART interface to operate as IrDA protocol by writing ‘10’ to the MODE (bits [25:24]) of the SCB_UART_CTRL register.
3. Enable the Median filter on the input interface line by writing ‘1’ to MEDIAN (bit 9) of the SCB_RX_CTRL register.
4. Configure the SCB as described in **“Enabling and initializing UART”** on page 183.

19.3.4 UART registers

The UART interface is controlled using a set of 32-bit registers listed in **Table 19-18**. For more information on these registers, see the PSoC™ 4 HV PA *Registers TRM*.

Table 19-18. UART registers

Register name	Operation
SCB_CTRL	Enables the SCB; selects the type of serial interface (SPI, UART, I^{2}C)
SCB_UART_CTRL	Used to select the sub-modes of UART (standard UART, SmartCard, IrDA), also used for local loop back control.
SCB_UART_RX_STATUS	Used to specify the BR_COUNTER value that determines the bit period. This is used to set the accuracy of the SCB clock. This value provides more granularity than the OVS bit in SCB_CTRL register.
SCB_UART_TX_CTRL	Used to specify the number of stop bits, enable parity, select the type of parity, and enable retransmission on NACK.
SCB_UART_RX_CTRL	Performs same function as SCB_UART_TX_CTRL but is also used for enabling multi processor mode, LIN mode drop on parity error, and drop on frame error.
SCB_TX_CTRL	Used to specify the data frame width and to specify whether MSB or LSB is the first bit in transmission.
SCB_RX_CTRL	Performs the same function as that of the SCB_TX_CTRL register, but for the receiver. Also decides whether a median filter is to be used on the input interface lines.
SCB_UART_FLOW_CONTROL	Configures flow control for UART transmitter.

19.3.5 UART interrupts

The UART supports both internal and external interrupt requests. The internal interrupt events are listed in this section. Custom ISRs can be used by connecting the external interrupt component to the interrupt output of the UART component (with external interrupts enabled).

The UART predefined interrupts can be classified as TX interrupts and RX interrupts. The TX interrupt output is the logical OR of the group of all possible TX interrupt sources. This signal goes high when any of the enabled TX interrupt sources is true. The RX interrupt output is the logical OR of the group of all possible RX interrupt sources. This signal goes high when any of the enabled Rx interrupt sources is true. The UART provides interrupts on the following events:

- TX
  - TX FIFO has less entries than the value specified by TRIGGER_LEVEL in SCB_TX_FIFO_CTRL
  - TX FIFO is not full
  - TX FIFO is empty
  - TX FIFO overflow
  - TX FIFO underflow
  - TX received a NACK in SmartCard mode
  - TX done
  - Arbitration lost (in LIN or SmartCard modes)
- RX
  - RX FIFO has less entries than the value specified by TRIGGER_LEVEL in SCB_RX_FIFO_CTRL
  - RX FIFO is full
  - RX FIFO is not empty
  - RX FIFO overflow
  - RX FIFO underflow
  - Frame error in received data frame
  - Parity error in received data frame
  - LIN baud rate detection is completed
  - LIN break detection is successful

19.3.6 Enabling and initializing UART

The UART must be programmed in the following order:

1. Program protocol specific information using the SCB_UART_CTRL register, according to **Table 19-19**. This includes selecting the submodes of the protocol, transmitter-receiver functionality, and so on.
2. Program the generic transmitter and receiver information using the SCB_TX_CTRL and SCB_RX_CTRL registers, as shown in **Table 19-21**.
   a) Specify the data frame width.
   b) Specify whether MSB or LSB is the first bit to be transmitted or received.
3. Program the transmitter and receiver FIFOs using the SCB_TX_FIFO_CTRL and SCB_RX_FIFO_CTRL registers respectively, as shown in **Table 19-21**.
   a) Set the trigger level.
   b) Clear the transmitter and receiver FIFO and Shift registers.
   c) Freeze the TX and RX FIFOs.
4. Program the SCB_CTRL register to enable the SCB block. Also select the mode of operation (**Table 19-22**).
5. Enable the block (write a '1' to the ENABLED bit of the SCB_CTRL register). After the block is enabled, control bits should not be changed. Changes should be made after disabling the block; for example, to modify the operation mode (from SmartCard to IrDA). The change takes effect only after the block is re-enabled. Note that re-enabling the block causes re-initialization and the associated state is lost (for example FIFO content).

Table 19-19. SCB_UART_CTRL Register

Bits	Name	Value	Description
[25:24]	MODE	00	Standard UART
		01	SmartCard
		10	IrDA
		11	Reserved
16	LOOP_BACK	Loop back control. This allows a SCB UART transmitter to communicate with its receiver counterpart.	

Table 19-20. SCB_TX_CTRL/SCB_RX_CTRL Registers

Bits	Name	Description
[3:0]	DATA_ WIDTH	'DATA_WIDTH + 1' is the no. of bits in the transmitted or received data frame. The valid range is [3, 15]. This does not include start, stop, and parity bits.
8	MSB_FIRST	1 = MSB first 0 = LSB first
9	MEDIAN	This is for SCB_RX_CTRL only. Decides whether a digital three-tap median filter is applied on the input interface lines. This filter should reduce susceptibility to errors, but it requires higher oversampling values. For the UART IrDA mode, this should always be '1'. 1 = Enabled 0 = Disabled

Table 19-21. SCB_TX_FIFO_CTRL/SCB_RX_FIFO_CTRL Registers

Bits	Name	Description
[3:0]	TRIGGER_LEVEL	Trigger level. When the transmitter FIFO has less entries or receiver FIFO has more entries than the value of this field, a transmitter or receiver trigger event is generated in the respective case.
16	CLEAR	When '1', the transmitter or receiver FIFO and the shift registers are cleared/invalidated.
17	FREEZE	When '1', hardware reads/writes to the transmitter or receiver FIFO have no effect. Freeze will not advance the TX or RX FIFO read/write pointer.

Table 19-22. SCB_CTRL Register

Bits	Name	Value	Description
[25:24]	MODE	00	I^{2}C mode
		01	SPI mode
		10	UART mode
		11	Reserved
31	ENABLED	0	SCB block disabled
		1	SCB block enabled

19.4 Inter integrated circuit (I^{2}C)

This section explains the I^{2}C implementation in PSoC™. For more information on the I^{2}C protocol specification, refer to the I^{2}C-bus specification available on the NXP website.

19.4.1 Features

This block supports the following features:

- Master, slave, and master/slave mode
- Slow-mode (50 Kbps), standard-mode (100 Kbps), fast-mode (400 Kbps), and fast-mode plus (1000 Kbps) data-rates
- 7- or 10-bit slave addressing (10-bit addressing requires firmware support)
- Clock stretching and collision detection
- Programmable oversampling of I^{2}C clock signal (SCL)
- Error reduction using an digital median filter on the input path of the I^{2}C data signal (SDA)
- Glitch-free signal transmission with an analog glitch filter
- Interrupt or polling CPU interface

19.4.2 General description

Figure 19-27 illustrates an example of an I^{2}C communication network.

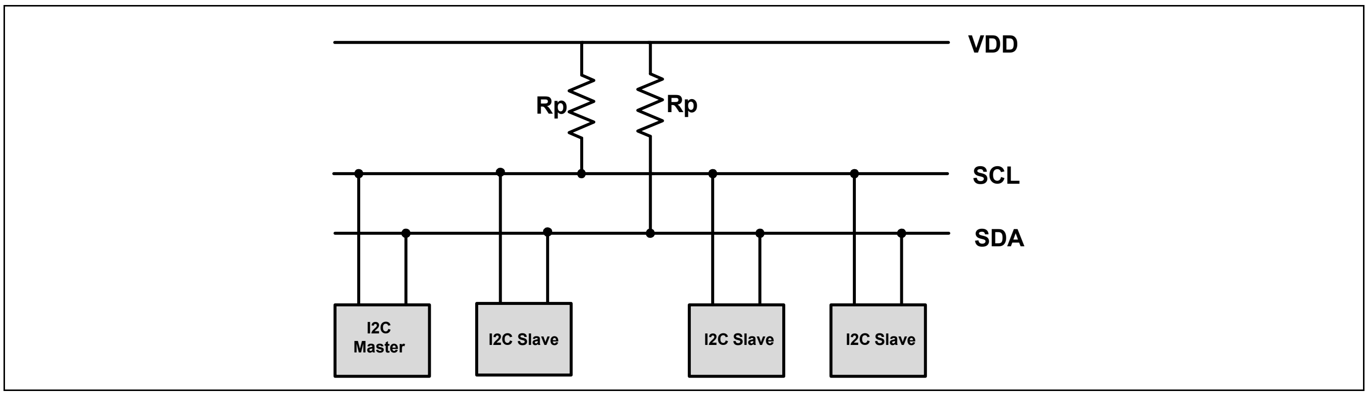


Figure 19-27. I^{2}C Interface block diagram

The standard I^{2}C bus is a two wire interface with the following lines:

- Serial Data (SDA)
- Serial Clock (SCL)

I^{2}C devices are connected to these lines using open collector or open-drain output stages, with pull-up resistors (Rp). A simple master/slave relationship exists between devices. Masters and slaves can operate as either transmitter or receiver. Each slave device connected to the bus is software addressable by a unique 7-bit address. PSoC™ 4 HV PA also supports 10-bit address matching for I^{2}C with firmware support.

19.4.3 Terms and definitions

Table 19-23 explains the commonly used terms in an I^2C communication network.

Table 19-23. Definition of I^2C Bus Terminology

Term	Description
Transmitter	The device that sends data to the bus.
Receiver	The device that receives data from the bus.
Master	The device that initiates a transfer, generates clock signals, and terminates a transfer.
Slave	The device addressed by a master.
Multi-master	More than one master can attempt to control the bus at the same time without corrupting the message.
Arbitration	Procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is not corrupted.
Synchronization	Procedure to synchronize the clock signals of two or more devices.

19.4.3.1 Clock stretching

When a slave device is not yet ready to process data, it may drive a ‘0’ on the SCL line to hold it down. Due to the implementation of the I/O signal interface, the SCL line value will be ‘0’, independent of the values that any other master or slave may be driving on the SCL line. This is known as clock stretching and is the only situation in which a slave drives the SCL line. The master device monitors the SCL line and detects it when it cannot generate a positive clock pulse (‘1’) on the SCL line. It then reacts by delaying the generation of a positive edge on the SCL line, effectively synchronizing with the slave device that is stretching the clock.

19.4.3.2 Bus arbitration

The I^2C protocol is a multi-master, multi-slave interface. Bus arbitration is implemented on master devices by monitoring the SDA line. Bus collisions are detected when the master observes an SDA line value that is not the same as the value it is driving on the SDA line. For example, when master 1 is driving the value ‘1’ on the SDA line and master 2 is driving the value ‘0’ on the SDA line, the actual line value will be ‘0’ due to the implementation of the I/O signal interface. Master 1 detects the inconsistency and loses control of the bus. Master 2 does not detect any inconsistency and keeps control of the bus.

19.4.4 I^{2}C modes of operation

I^{2}C is a synchronous single master, multi-master, multi-slave serial interface. Devices operate in either master mode, slave mode, or master/slave mode. In master/slave mode, the device switches from master to slave mode when it is addressed. Only a single master may be active during a data transfer. The active master is responsible for driving the clock on the SCL line. **Table 19-24** illustrates the I^{2}C modes of operation.

Table 19-24. I^{2}C Modes

Mode	Description
Slave	Slave only operation (default)
Master	Master only operation
Multi-master	Supports more than one master on the bus
Multi-master-slave	Simultaneous slave and multi-master operation

Data transfer through the I^{2}C bus follows a specific format. **Table 19-25** lists some common bus events that are part of an I^{2}C data transfer. The **"Write transfer"** on page 188 and **"Read transfer"** on page 189 sections explain the I^{2}C bus bit format during data transfer.

Table 19-25. I^{2}C bus events terminology

Bus Event	Description
START	A HIGH to LOW transition on the SDA line while SCL is HIGH.
STOP	A LOW to HIGH transition on the SDA line while SCL is HIGH.
ACK	The receiver pulls the SDA line LOW and it remains LOW during the HIGH period of the clock pulse, after the transmitter transmits each byte. This indicates to the transmitter that the receiver received the byte properly.
NACK	The receiver does not pull the SDA line LOW and it remains HIGH during the HIGH period of clock pulse after the transmitter transmits each byte. This indicates to the transmitter that the receiver received the byte properly.
Repeated START	START condition generated by master at the end of a transfer instead of a STOP condition.
DATA	SDA status change while SCL is low (data changing), and no change while SCL is high (data valid).

When operating in multi-master mode, the bus should always be checked to see if it is busy; another master may already be communicating with a slave. In this case, the master must wait until the current operation is complete before issuing a START signal (see **Table 19-25**, **Figure 19-28**, and **Figure 19-29**). The master looks for a STOP signal as an indicator that it can start its data transmission.

When operating in multi-master-slave mode, if the master loses arbitration during data transmission, the hardware reverts to slave mode and the received byte generates a slave address interrupt, so that the device is ready to respond to any other master on the bus. With all of these modes, there are two types of transfer - read and write. In write transfer, the master sends data to slave; in read transfer, the master receives data from slave. Write and read transfer examples are available in **"Master mode transfer examples"** on page 199, **"Slave mode transfer examples"** on page 201, and **"Multi-Master mode transfer example"** on page 205.

19.4.4.1 Write transfer

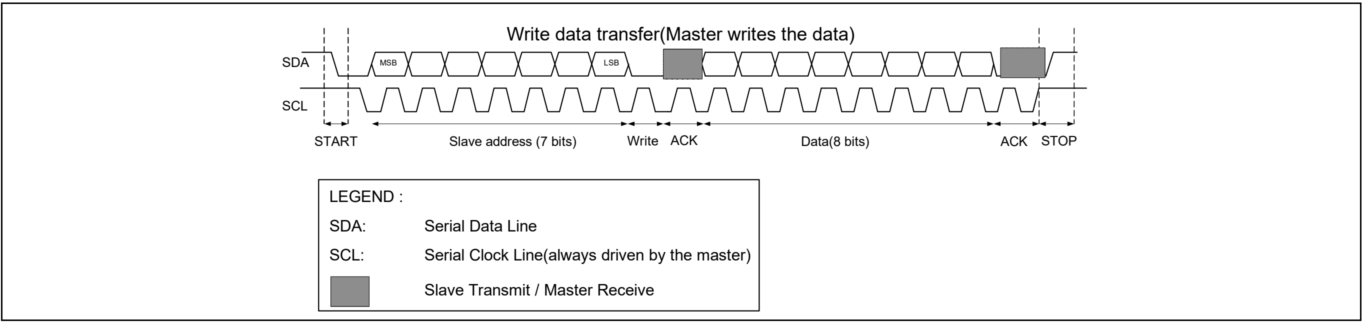

Write data transfer(Master writes the data)
SDA
SCL
MSB
LSB
START
Slave address (7 bits)
Write
ACK
Data(8 bits)
ACK
STOP
LEGEND :
SDA: Serial Data Line
SCL: Serial Clock Line(always driven by the master)
Slave Transmit / Master Receive

Figure 19-28. Master write data transfer

- A typical write transfer begins with the master generating a START condition on the I^{2}C bus. The master then writes a 7-bit I^{2}C slave address and a write indicator ('0') after the START condition. The addressed slave transmits an acknowledgement byte by pulling the data line low during the ninth bit time.
- If the slave address does not match any of the slave devices or if the addressed device does not want to acknowledge the request, it transmits a no acknowledgement (NACK) by not pulling the SDA line low. The absence of an acknowledgement, results in an SDA line value of '1' due to the pull-up resistor implementation.
- If no acknowledgement is transmitted by the slave, the master may end the write transfer with a STOP event. The master can also generate a repeated START condition for a retry attempt.
- The master may transmit data to the bus if it receives an acknowledgement. The addressed slave transmits an acknowledgement to confirm the receipt of every byte of data written. Upon receipt of this acknowledgement, the master may transmit another data byte.
- When the transfer is complete, the master generates a STOP condition.

19.4.4.2 Read transfer

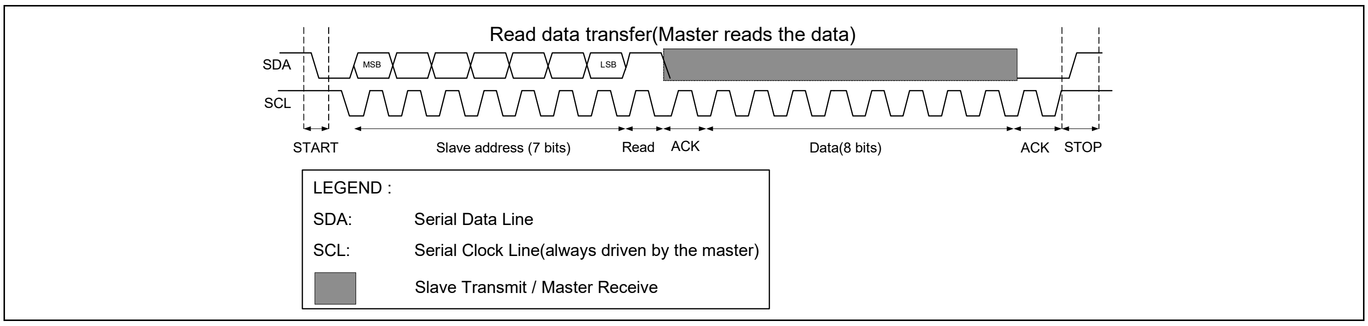

Read data transfer(Master reads the data)
SDA
SCL
MSB
LSB
START
Slave address (7 bits)
Read
ACK
Data(8 bits)
ACK
STOP
LEGEND :
SDA: Serial Data Line
SCL: Serial Clock Line(always driven by the master)
Slave Transmit / Master Receive

Figure 19-29. Master read data transfer

- A typical read transfer begins with the master generating a START condition on the I^{2}C bus. The master then writes a 7-bit I^{2}C slave address and a read indicator (‘1’) after the START condition. The addressed slave transmits an acknowledgement by pulling the data line low during the ninth bit time.
- If the slave address does not match with that of the connected slave device or if the addressed device does not want to acknowledge the request, a no acknowledgement (NACK) is transmitted by not pulling the SDA line low. The absence of an acknowledgement, results in an SDA line value of ‘1’ due to the pull-up resistor implementation.
- If no acknowledgement is transmitted by the slave, the master may end the read transfer with a STOP event. The master can also generate a repeated START condition for a retry attempt.
- If the slave acknowledges the address, it starts transmitting data after the acknowledgement signal. The master transmits an acknowledgement to confirm the receipt of each data byte sent by the slave. Upon receipt of this acknowledgement, the addressed slave may transmit another data byte.
- The master can send a NACK signal to the slave to stop the slave from sending data bytes. This completes the read transfer.
- When the transfer is complete, the master generates a STOP condition.

19.4.5 Easy I2C (EZI2C) protocol

The Easy I2C (EZI2C) protocol is a unique communication scheme built on top of the I^2C protocol by Infineon. It uses a software wrapper around the standard I^2C protocol to communicate to an I^2C slave using indexed memory transfers. This removes the need for CPU intervention at the level of individual frames.

The EZI2C protocol defines an 8-bit address that indexes a memory array (8-bit wide 32 locations) located on the slave device. Five lower bits of the EZ address are used to address these 32 locations. The number of bytes transferred to or from the EZI2C memory array can be found by comparing the EZ address at the START event and the EZ address at the STOP event.

Note: The I^2C block has a hardware FIFO memory, which is 16 bits wide and 16 locations deep with byte write enable. The access methods for EZ and non-EZ functions are different. In non-EZ mode, the FIFO is split into TXFIFO and RXFIFO. Each has 16-bit wide eight locations. In EZ mode, the FIFO is used as a single memory unit with 8-bit wide 32 locations.

EZI2C has two types of transfers: a data write from the master to an addressed slave memory location, and a read by the master from an addressed slave memory location.

19.4.5.1 Memory array write

An EZ write to a memory array index is by means of an I^2C write transfer. The first transmitted write data is used to send an EZ address from the master to the slave. The five lowest significant bits of the write data are used as the “new” EZ address at the slave. Any additional write data elements in the write transfer are bytes that are written to the memory array. The EZ address is automatically incremented by the slave as bytes are written into the memory array. If the number of continuous data bytes written to the EZI2C buffer exceeds EZI2C buffer boundary, it overwrites the last location for every subsequent byte.

19.4.5.2 Memory array read

An EZ read from a memory array index is by means of an I^2C read transfer. The EZ read relies on an earlier EZ write to have set the EZ address at the slave. The first received read data is the byte from the memory array at the EZ address memory location. The EZ address is automatically incremented as bytes are read from the memory array. The address wraps around to zero when the final memory location is reached.

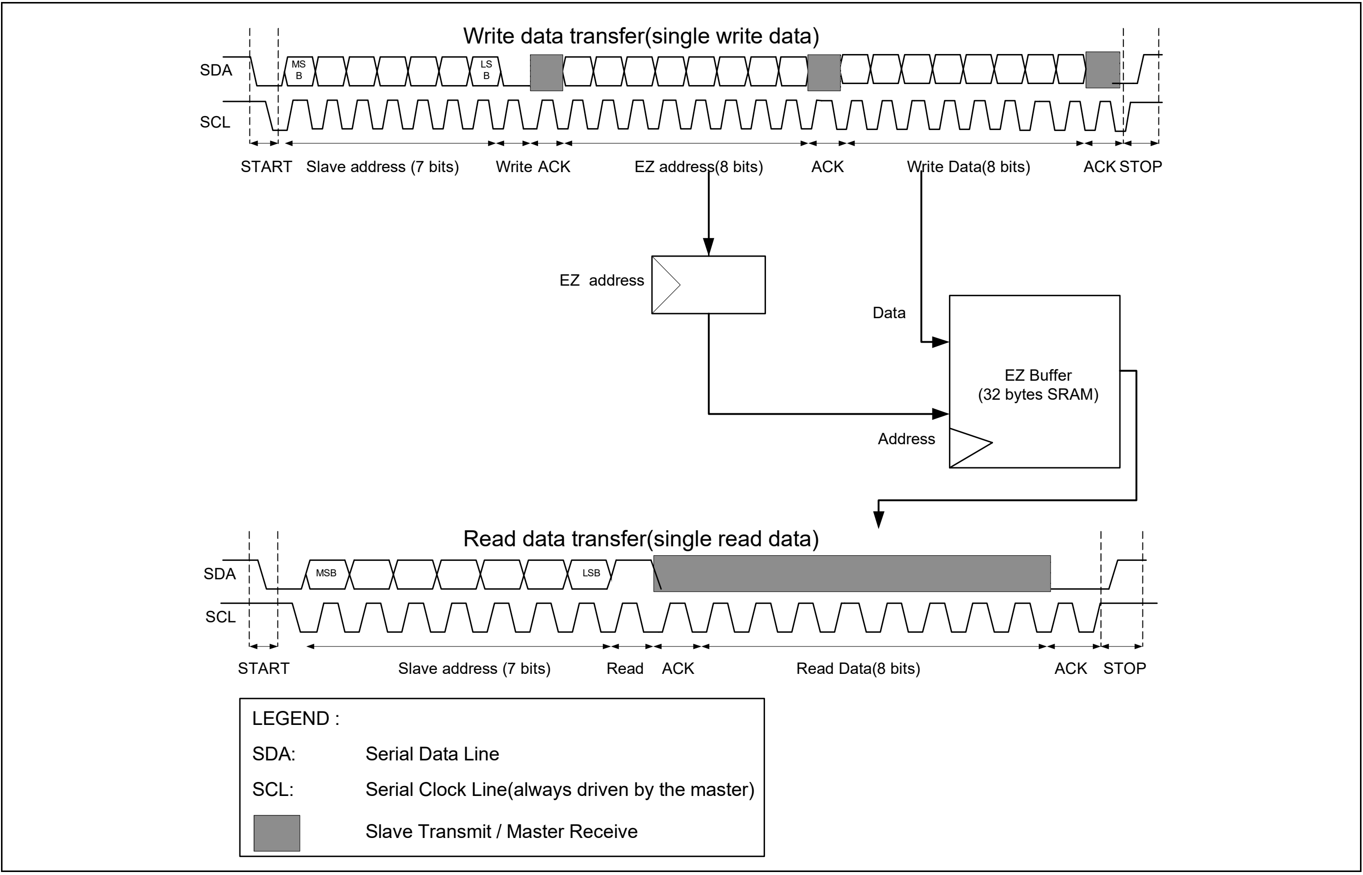


Figure 19-30. EZI2C write and read data transfer

19.4.6 I^{2}C registers

The I^{2}C interface is controlled by reading and writing a set of configuration, control, and status registers, as listed in **Table 19-26**.

Table 19-26. I^{2}C Registers

Register	Function
SCB_CTRL	Enables the I^{2}C block and selects the type of serial interface (SPI, UART, I^{2}C). Also used to select internally and externally clocked operation and EZ and non-EZ modes of operation.
SCB_I2C_CTRL	Selects the mode (master, slave) and sends an ACK or NACK signal based on receiver FIFO status.
SCB_I2C_STATUS	Indicates bus busy status detection, read/write transfer status of the slave/master, and stores the EZ slave address.
SCB_I2C_M_CMD	Enables the master to generate START, STOP, and ACK/NACK signals.
SCB_I2C_S_CMD	Enables the slave to generate ACK/NACK signals.
SCB_STATUS	Indicates whether the externally clocked logic is using the EZ memory. This bit can be used by software to determine whether it is safe to issue a software access to the EZ memory.
SCB_I2C_CFG	Configures filters, which remove glitches from the SDA and SCL lines.
SCB_TX_CTRL	Specifies the data frame width; also used to specify whether MSB or LSB is the first bit in transmission.
SCB_TX_FIFO_CTRL	Specifies the trigger level, clearing of the transmitter FIFO and shift registers, and FREEZE operation of the transmitter FIFO.
SCB_TX_FIFO_STATUS	Indicates the number of bytes stored in the transmitter FIFO, the location from which a data frame is read by the hardware (read pointer), the location from which a new data frame is written (write pointer), and decides if the transmitter FIFO holds the valid data.
SCB_TX_FIFO_WR	Holds the data frame written into the transmitter FIFO. Behavior is similar to that of a PUSH operation.
SCB_RX_CTRL	Performs the same function as that of the SCB_TX_CTRL register, but for the receiver. Also decides whether a median filter is to be used on the input interface lines.
SCB_RX_FIFO_CTRL	Performs the same function as that of the SCB_TX_FIFO_CTRL register, but for the receiver.
SCB_RX_FIFO_STATUS	Performs the same function as that of the SCB_TX_FIFO_STATUS register, but for the receiver.
SCB_RX_FIFO_RD	Holds the data read from the receiver FIFO. Reading a data frame removes the data frame from the FIFO; behavior is similar to that of a POP operation. This register has a side effect when read by software: a data frame is removed from the FIFO.
SCB_RX_FIFO_RD_SILENT	Holds the data read from the receiver FIFO. Reading a data frame does not remove the data frame from the FIFO; behavior is similar to that of a PEEK operation.
SCB_RX_MATCH	Stores slave device address and is also used as slave device address MASK.
SCB_EZ_DATA	Holds the data in an EZ memory location.

Note: Detailed descriptions of the I^{2}C register bits are available in the PSoC™ 4 HV PA Registers TRM.

19.4.7 I^{2}C interrupts

The fixed-function I^{2}C block generates interrupts for the following conditions.

- I^{2}C Master
  - I^{2}C master lost arbitration
  - I^{2}C master received NACK
  - I^{2}C master received ACK
  - I^{2}C master sent STOP
  - I^{2}C bus error (unexpected stop/start condition detected)
- I^{2}C Slave
  - I^{2}C slave lost arbitration
  - I^{2}C slave received NACK
  - I^{2}C slave received ACK
  - I^{2}C slave received STOP
  - I^{2}C slave received START
  - I^{2}C slave address matched
  - I^{2}C bus error (unexpected stop/start condition detected)
- TX
  - TX FIFO has less entries than the value specified by TRIGGER_LEVEL in SCB_TX_FIFO_CTRL
  - TX FIFO is not full
  - TX FIFO is empty
  - TX FIFO overflow
  - TX FIFO underflow
- RX
  - RX FIFO has less entries than the value specified by TRIGGER_LEVEL in SCB_RX_FIFO_CTRL
  - RX FIFO is full
  - RX FIFO is not empty
  - RX FIFO overflow
  - RX FIFO underflow
- I^{2}C Externally Clocked
  - Wake up request on address match
  - I^{2}C STOP detection at the end of each transfer
  - I^{2}C STOP detection at the end of a write transfer
  - I^{2}C STOP detection at the end of a read transfer

The I^{2}C interrupt signal is hard-wired to the Cortex-M0 NVIC and cannot be routed to external pins.

The interrupt output is the logical OR of the group of all possible interrupt sources. The interrupt is triggered when any of the enabled interrupt conditions are met. Interrupt status registers are used to determine the actual source of the interrupt. For more information on interrupt registers, see the PSoC™ 4 HV PA Registers TRM.

19.4.8 Enabling and initializing the I^{2}C

The following section describes the method to configure the I^{2}C block for standard (non-EZ) mode and EZI2C mode.

19.4.8.1 I^{2}C Standard (Non-EZ) mode configuration

The I^{2}C interface must be programmed in the following order.

1. Program protocol specific information using the SCB_I2C_CTRL register according to **Table 19-27**. This includes selecting master - slave functionality.
2. Program the generic transmitter and receiver information using the SCB_TX_CTRL and SCB_RX_CTRL registers, as shown in **Table 19-28**.
   a) Specify the data frame width.
   b) Specify that MSB is the first bit to be transmitted/received.
3. Program transmitter and receiver FIFO using the SCB_TX_FIFO_CTRL and SCB_RX_FIFO_CTRL registers, respectively, as shown in **Table 19-29**.
   a) Set the trigger level.
   b) Clear the transmitter and receiver FIFO and Shift registers.
4. Program the SCB_CTRL register to enable the I^{2}C block and select the I^{2}C mode. These register bits are shown in **Table 19-30**. For a complete description of the I^{2}C registers, see the PSoC™ 4 HV PA Registers TRM.

Table 19-27. SCB_I2C_CTRL Register

Bits	Name	Value	Description
30	SLAVE_MODE	1	Slave mode
31	MASTER_MODE	1	Master mode

Table 19-28. SCB_TX_CTRL/SCB_RX_CTRL Register

Bits	Name	Description
[3:0]	DATA_ WIDTH	'DATA_WIDTH + 1' is the number of bits in the transmitted or received data frame. For I^{2}C, this is always 7.
8	MSB_FIRST	1 = MSB first (this should always be true for I^{2}C)
		0 = LSB first
9	MEDIAN	This is for SCB_RX_CTRL only. Decides whether a digital three-tap median filter is applied on the input interface lines. This filter should reduce susceptibility to errors, but it requires higher oversampling values. 1 = Enabled 0 = Disabled

Table 19-29. SCB_TX_FIFO_CTRL/SCB_RX_FIFO_CTRL

Bits	Name	Description
[3:0]	TRIGGER_LEVEL	Trigger level. When the transmitter FIFO has less entries or the receiver FIFO has more entries than the value of this field, a transmitter or receiver trigger event is generated in the respective case.
16	CLEAR	When ‘1’, the transmitter or receiver FIFO and the shift registers are cleared.
17	FREEZE	When ‘1’, hardware reads/writes to the transmitter or receiver FIFO have no effect. Freeze does not advance the TX or RX FIFO read/write pointer.

Table 19-30. SCB_CTRL Registers

Bits	Name	Value	Description
[25:24]	MODE	00	I^2C mode
		01	SPI mode
		10	UART mode
		11	Reserved
31	ENABLED	0	SCB block disabled
		1	SCB block enabled

19.4.8.2 EZI2C mode configuration

To configure the I^2C block for EZI2C mode, set the following I^2C register bits

1. Select the EZI2C mode by writing ‘1’ to the EZ_MODE bit (bit 10) of the SCB_CTRL register.
2. Follow steps 2 to 4 mentioned in **“I2C Standard (Non-EZ) mode configuration”** on page 194.
3. Set the S_READY_ADDR_ACK (bit 12) and S_READY_DATA_ACK (bit 13) bits of the SCB_I2C_CTRL register.

19.4.9 Internal and external clock operation in I^{2}C

The I^{2}C block supports both internally and externally clocked operation for data-rate generation. Internally clocked operations use a clock signal derived from the PSoC™ system bus clock. Externally clocked operations use a clock provided by the user. Externally clocked operation allows limited functionality in the Deep Sleep power mode, in which on-chip clocks are not active. For more information on system clocking, see the **"Clocking system"** on page 93.

Externally clocked operation is limited to the following cases:

- Slave functionality.
- EZ functionality.

TX and RX FIFOs do not support externally clocked operation; therefore, it is not used for non-EZ functionality.

Internally and externally clocked operations are determined by two register fields of the SCB_CTRL register:

- **EC_AM_MODE (Externally Clocked Address Matching Mode):** Indicates whether I^{2}C address matching is internally ('0') or externally ('1') clocked.
- **EC_OP_MODE (Externally Clocked Operation Mode):** Indicates whether the rest of the protocol operation (besides I^{2}C address match) is internally ('0') or externally ('1') clocked. As mentioned earlier, externally clocked operation does not support non-EZ functionality.

These two register fields determine the functional behavior of I^{2}C. The register fields should be set based on the required behavior in Active, Sleep, and Deep Sleep system power modes. Improper setting may result in faulty behavior in certain power modes. **Table 19-31** and **Table 19-32** describe the settings for I^{2}C in EZ and non-EZ mode.

19.4.9.1 I²C Non-EZ mode of operation

Externally clocked operation is not supported for non-EZ functionality because there is no FIFO support for this mode. So, the EC_OP_MODE should always be set to '0' for non-EZ mode. However, EC_AM_MODE can be set to ‘0’ or ‘1’. **Table 19-31** gives an overview of the possibilities. The combination EC_AM_MODE = 0 and EC_OP_MODE = 1 is invalid and the block will not respond.

EC_AM_MODE is ‘0’ and EC_OP_MODE is ‘0’

This setting only works in Active and Sleep system power modes. All the functionality of the I²C is provided in the internally clocked domain.

EC_AM_MODE is ‘1’ and EC_OP_MODE is ‘0’

This setting works in Active, Sleep, and Deep Sleep system power modes. I²C address matching is performed by the externally clocked logic in Active, Sleep, and Deep Sleep system power modes. When the externally clocked logic matches the address, it sets a wakeup interrupt cause bit, which can be used to generate an interrupt to wakeup the CPU.

Table 19-31. I²C operation in non-EZ Mode

I²C (Non-EZ) Mode				
System Power Mode	**EC_OP_MODE = 0**		**EC_OP_MODE = 1**	
	EC_AM_MODE = 0	**EC_AM_MODE = 1**	**EC_AM_MODE = 0**	**EC_AM_MODE = 1**
Active and Sleep	Address match using internal clock. Operation using internal clock.	Address match using external clock. Operation using internal clock.	Not supported	
Deep Sleep	Not supported	Address match using external clock. Operation using internal clock.		

- In Active system power mode, the CPU is active and the wakeup interrupt cause is disabled (associated MASK bit is ‘0’). The externally clocked logic takes care of the address matching and the internally locked logic takes care of the rest of the I²C transfer.
- In the Sleep mode, wakeup interrupt cause can be either enabled or disabled based on the application. The remaining operations are similar to the Active mode.
- In the Deep Sleep mode, the CPU is shut down and will wake up on I²C activity if the wakeup interrupt cause is enabled. CPU wakeup up takes time and the ongoing I²C transfer is either negatively acknowledged (NACK) or the clock is stretched. In the case of a NACK, the internally clocked logic takes care of the first I²C transfer after it wakes up. For clock stretching, the internally clocked logic takes care of the ongoing/stretched transfer when it wakes up. The register bit S_NOT_READY_ADDR_NACK (bit 14) of the SCB_I2C_CTRL register determines whether the externally clocked logic performs a negative acknowledge (‘1’) or clock stretch (‘0’).

19.4.9.2 I^{2}C EZ operation mode

EZ mode has three possible settings. EC_AM_MODE can be set to '0' or '1' when EC_OP_MODE is '0' and EC_AM_MODE must be set to '1' when EC_OP_MODE is '1'. **Table 19-32** gives an overview of the possibilities. The gray cells indicate a possible, yet not recommended setting because it involves a switch from the externally clocked logic (slave selection) to the internally clocked logic (rest of the operation). The combination EC_AM_MODE = 0 and EC_OP_MODE = 1 is invalid and the block will not respond.

Table 19-32. I^{2}C operation in EZ Mode

I^{2}C, EZ Mode				
System Power Mode	**EC_OP_MODE= 0**		**EC_OP_MODE = 1**	
	EC_AM_MODE = 0	**EC_AM_MODE = 1**	**EC_AM_MODE = 0**	**EC_AM_MODE = 1**
Active and Sleep	Address match using internal clock Operation using internal clock	Address match using external clock Operation using internal clock	Invalid	Address match using external clock Operation using external clock
Deep Sleep	Not supported	Address match using external clock Operation using internal clock		Address match using external clock Operation using external clock

- **EC_AM_MODE is '0' and EC_OP_MODE is '0'**. This setting only works in Active and Sleep system power modes.
- **EC_AM_MODE is '1' and EC_OP_MODE is '0'**. This setting works same as I^{2}C non-EZ mode.
- **EC_AM_MODE is '1' and EC_OP_MODE is '1'**. This setting works in Active and Deep Sleep system power modes.

The I^{2}C block's functionality is provided in the externally clocked domain. Note that this setting results in externally clocked accesses to the block's SRAM. These accesses may conflict with internally clocked accesses from the device. This may cause wait states or bus errors. The field FIFO_BLOCK (bit 17) of the SCB_CTRL register determines whether wait states ('1') or bus errors ('0') are generated.

19.4.10 Wake up from Sleep

The system wakes up from Sleep or Deep Sleep system power modes when an I^{2}C address match occurs. The fixed-function I^{2}C block performs either of two actions after address match: Address ACK or Address NACK.

Address ACK - The I^{2}C slave executes clock stretching and waits until the device wakes up and ACKs the address.

Address NACK - The I^{2}C slave NACKs the address immediately. The master must poll the slave again after the device wakeup time is passed. This option is only valid in the slave or multi-master-slave modes.

Note: The interrupt bit WAKE_UP (bit 0) of the SCB_INTR_I2C_EC register must be enabled for the I^{2}C to wake up the device on slave address match while switching to the Sleep mode.

Note: If the device is configured in I^{2}C slave mode, the clock to the SCB should be disabled when entering Deep Sleep power mode; enable the clock when waking up from Deep Sleep mode.

19.4.11 Master mode transfer examples

Master mode transmits or receives data.

19.4.11.1 Master transmit

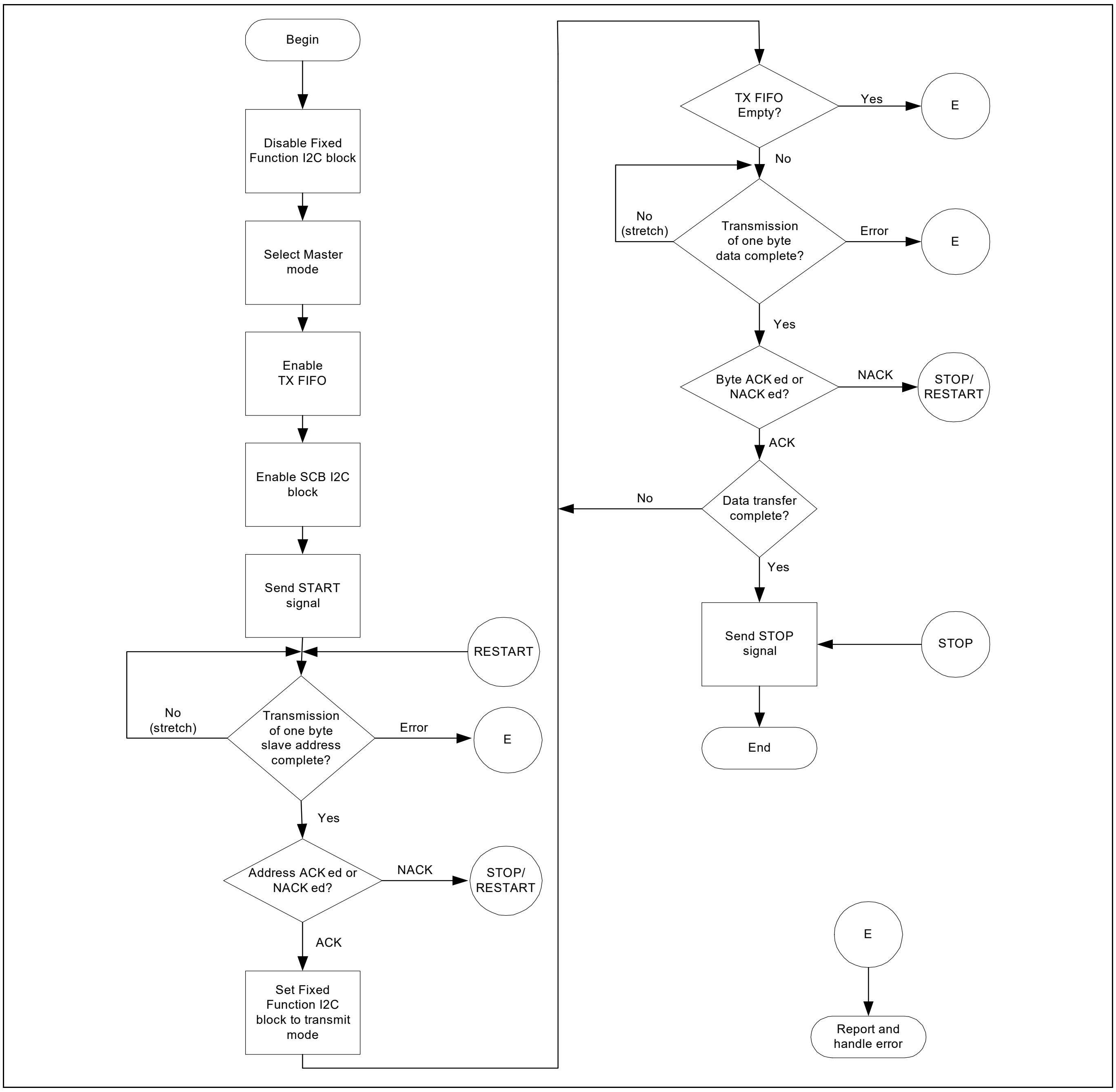


Figure 19-31. Single Master Mode Write operation flow chart

19.4.11.2 Master receive

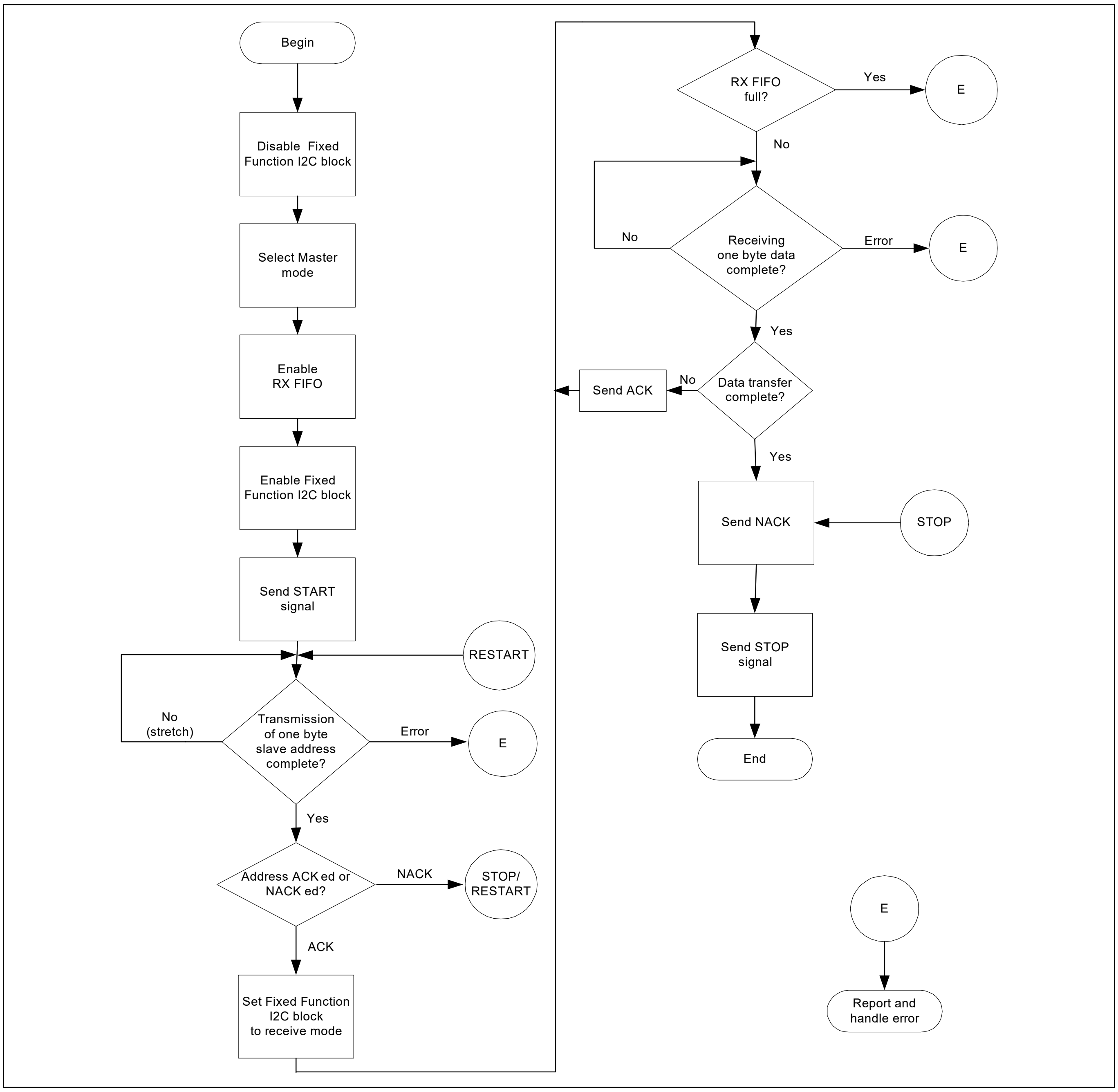


Figure 19-32. Single Master Mode Read operation flow chart

19.4.12 Slave mode transfer examples

Slave mode transmits or receives data.

19.4.12.1 Slave transmit

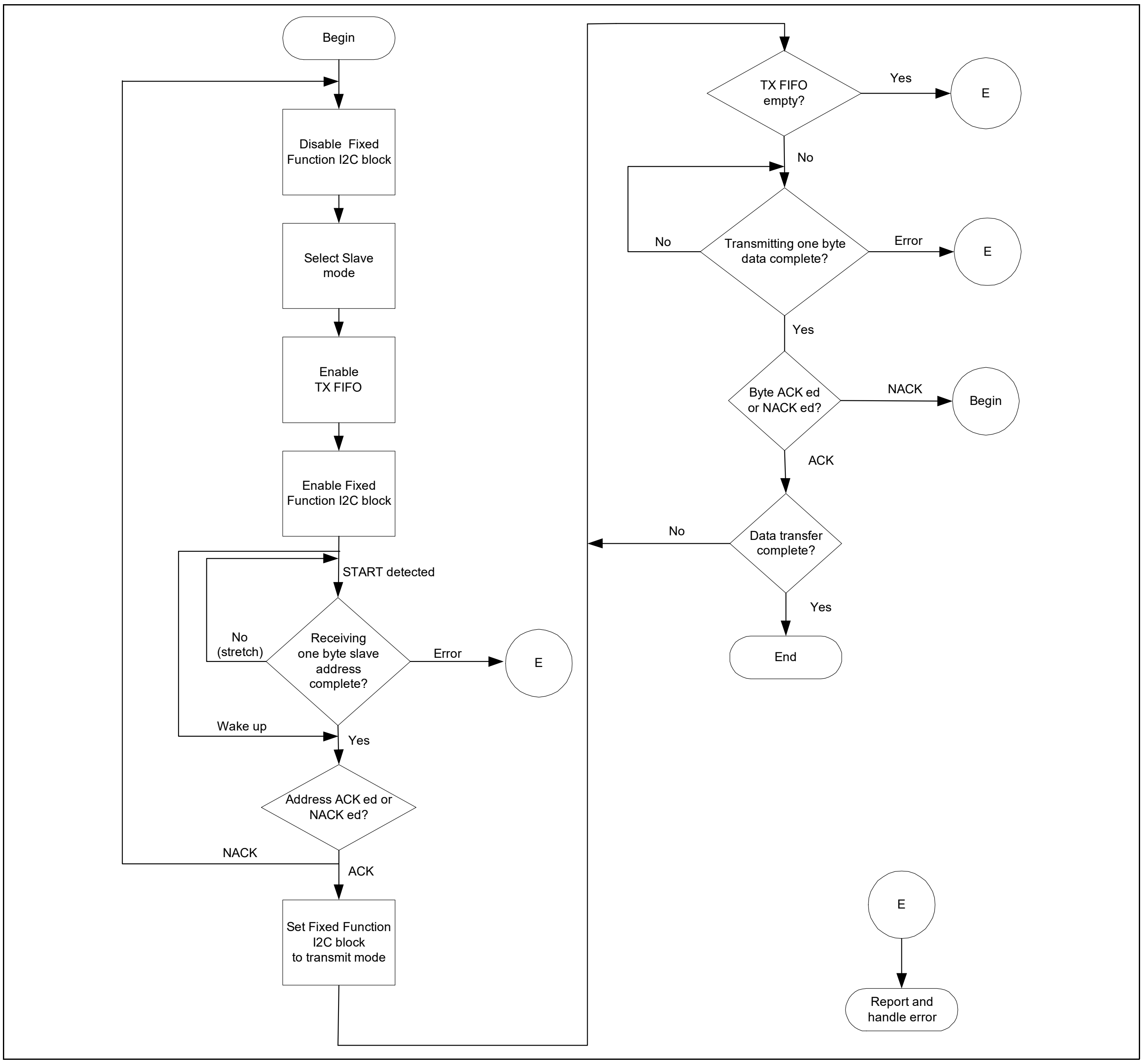

Begin
Disable Fixed Function I2C block
Select Slave mode
Enable TX FIFO
Enable Fixed Function I2C block
START detected
No (stretch)
Receiving one byte slave address complete?
Error
E
Wake up
Yes
Address ACK ed or NACK ed?
NACK
ACK
Set Fixed Function I2C block to transmit mode
TX FIFO empty?
Yes
E
No
No
Transmitting one byte data complete?
Error
E
Yes
Byte ACK ed or NACK ed?
NACK
Begin
ACK
No
Data transfer complete?
Yes
End
E
Report and handle error

Figure 19-33. Slave Mode Write operation flow chart

19.4.12.2 Slave receive

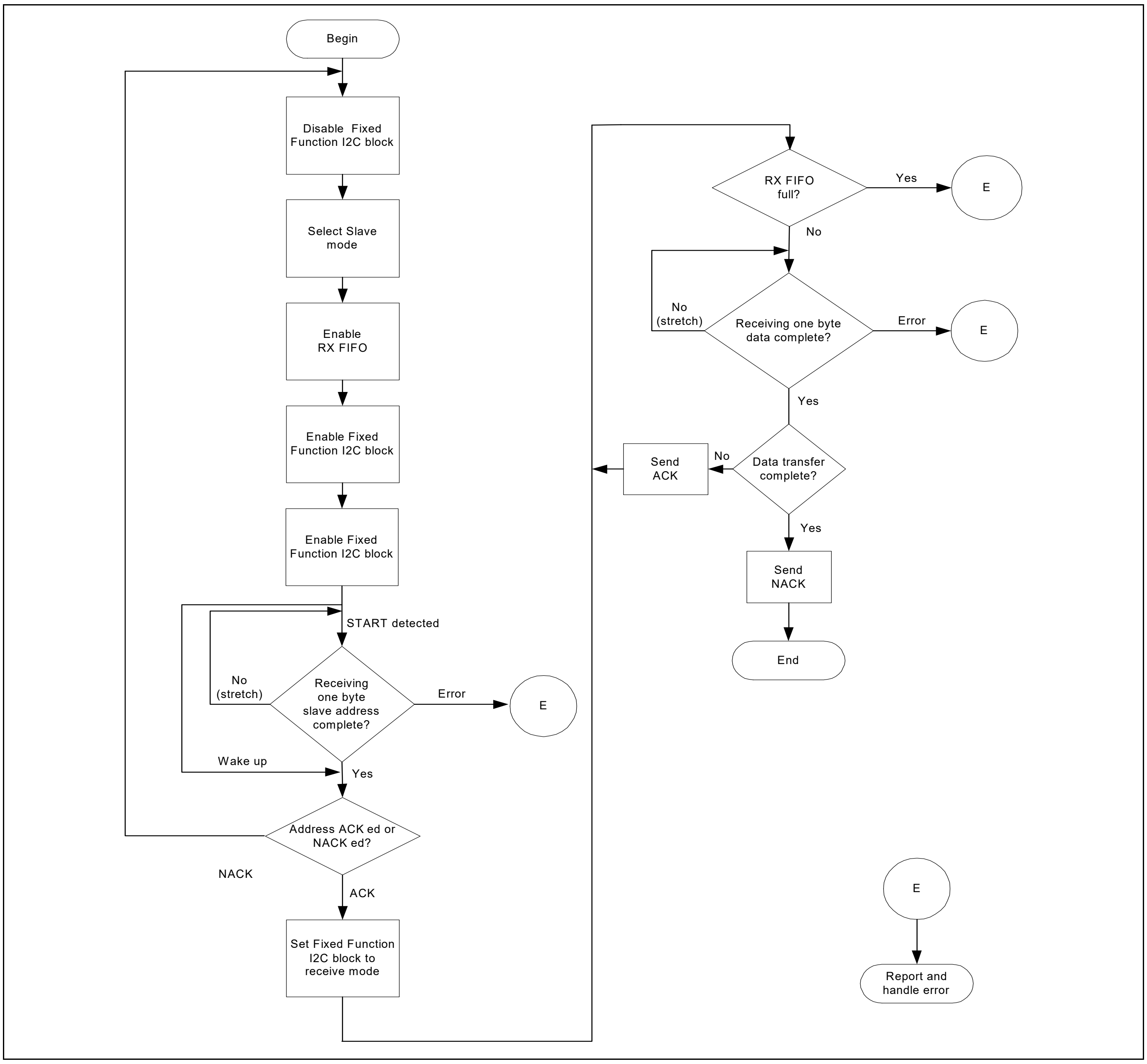

Begin
Disable Fixed Function I2C block
Select Slave mode
Enable RX FIFO
Enable Fixed Function I2C block
Enable Fixed Function I2C block
START detected
No (stretch)
Receiving one byte slave address complete?
Error
E
Wake up
Yes
Address ACK ed or NACK ed?
NACK
ACK
Set Fixed Function I2C block to receive mode
RX FIFO full?
Yes
E
No
No (stretch)
Receiving one byte data complete?
Error
E
Yes
Send ACK
No
Data transfer complete?
Yes
Send NACK
End
E
Report and handle error

Figure 19-34. Slave Mode Read operation flow chart

19.4.13 EZ Slave mode transfer example

The EZ Slave mode transmits or receives data.

19.4.13.1 EZ Slave transmit

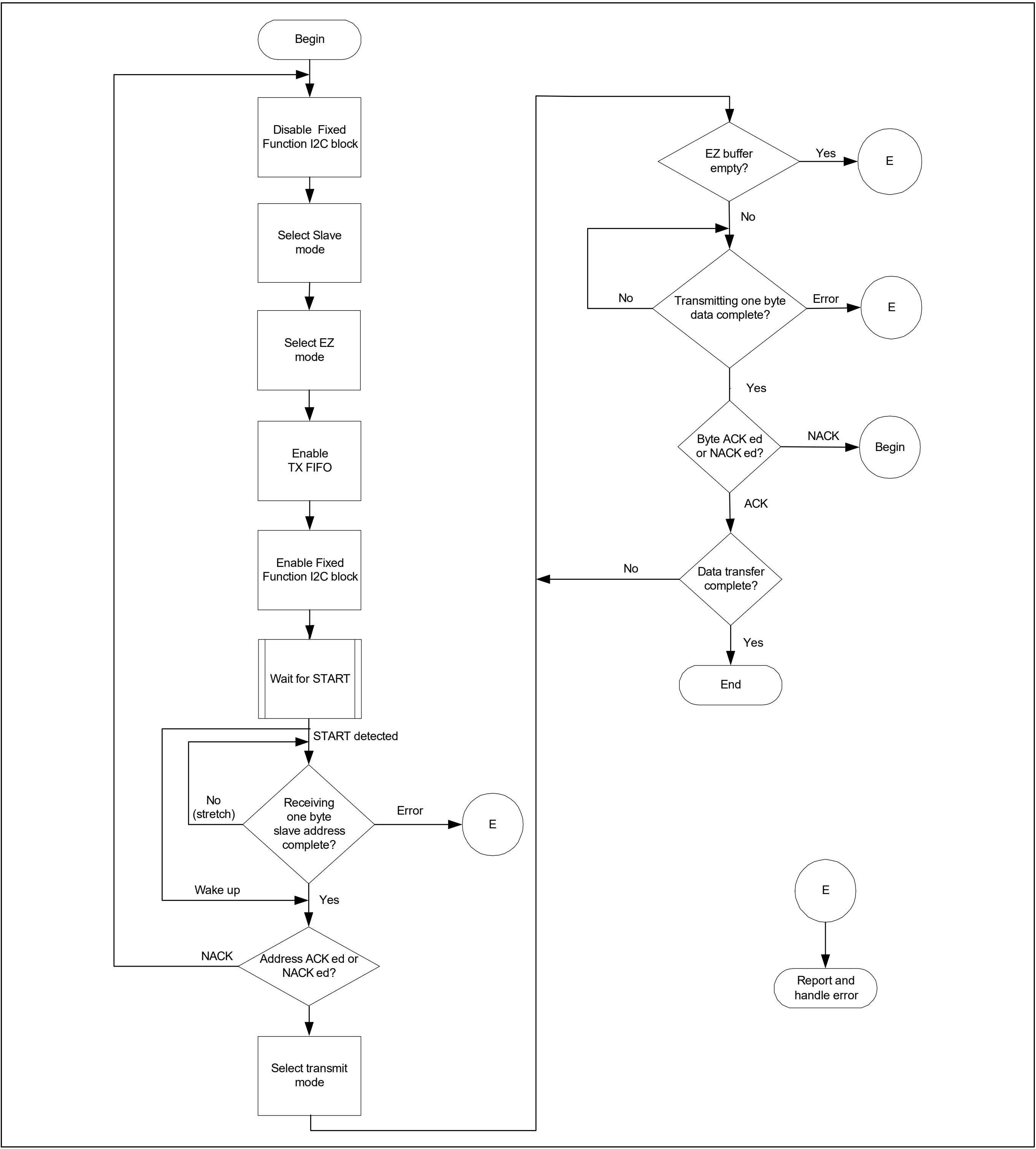


Figure 19-35. EZI2C Slave Mode Write operation flow chart

19.4.13.2 EZ Slave receive

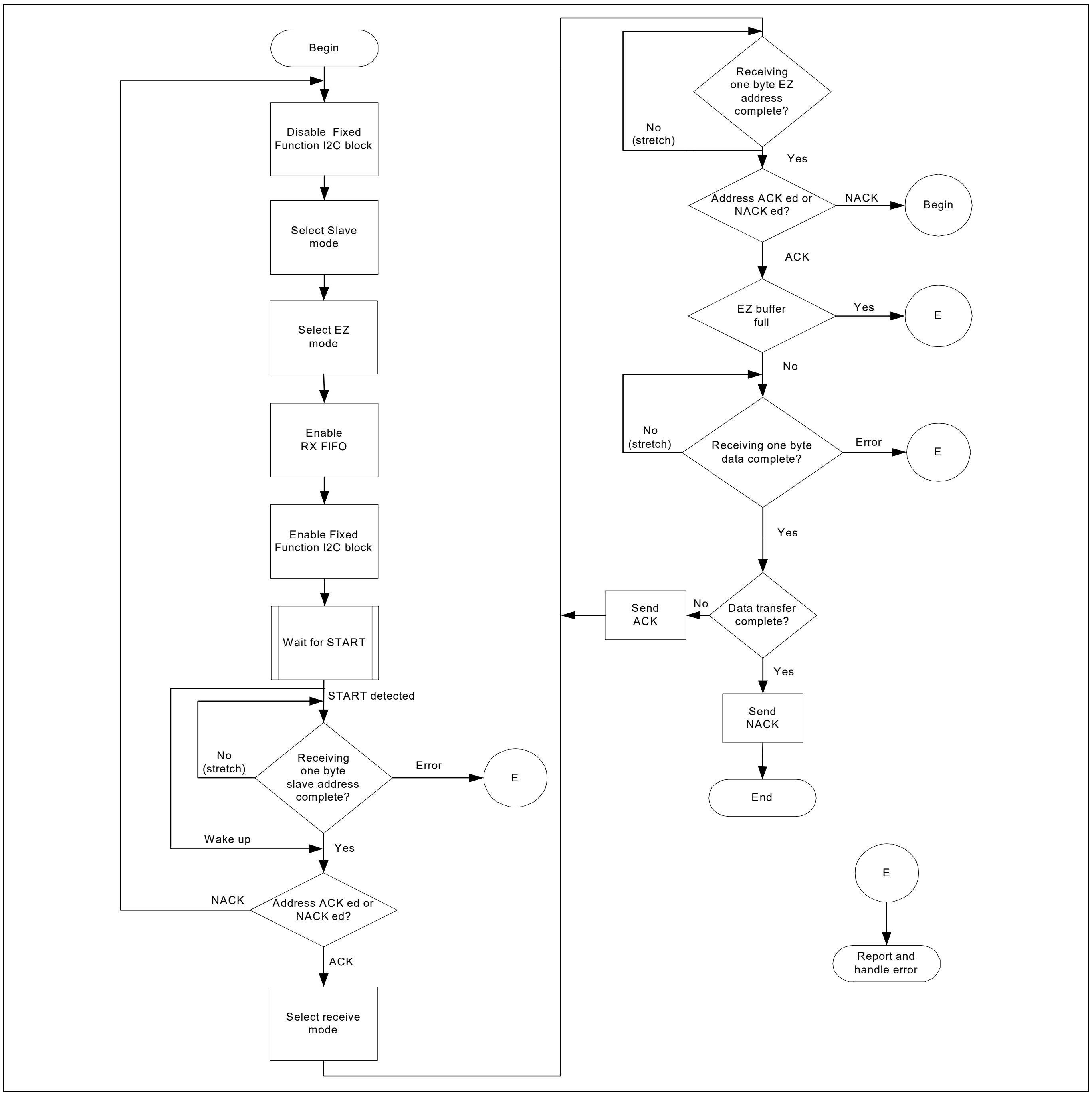


Figure 19-36. EZI2C Slave Mode Read operation flow chart

19.4.14 Multi-Master mode transfer example

In multi-master mode, data can be transferred with the slave mode enabled or not enabled.

19.4.14.1 Multi-Master - Slave not enabled

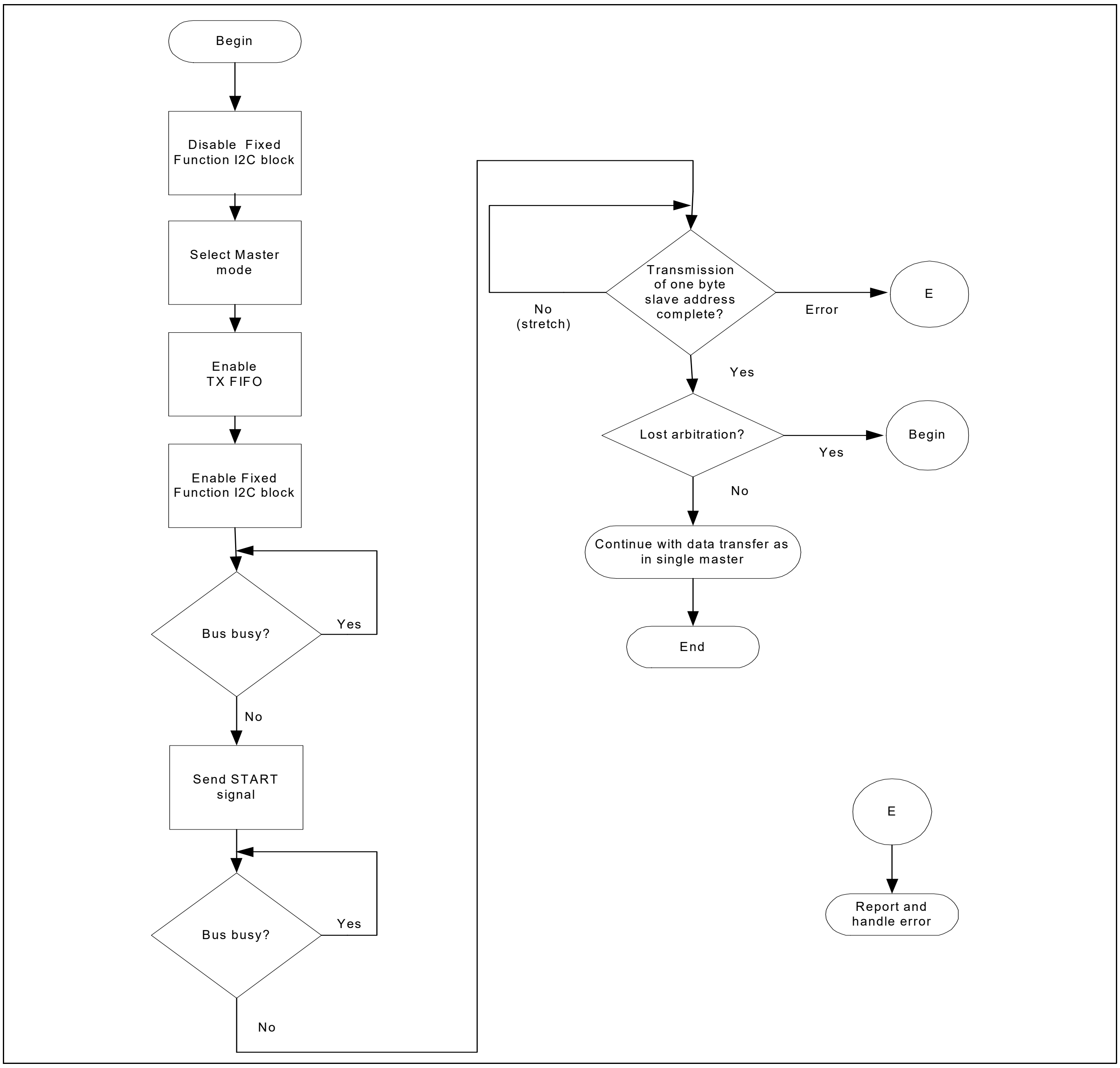

Begin
Disable Fixed Function I2C block
Select Master mode
Enable TX FIFO
Enable Fixed Function I2C block
Bus busy?
Yes
No
Send START signal
Bus busy?
Yes
No
Transmission of one byte slave address complete?
No (stretch)
Error
E
Yes
Lost arbitration?
Yes
Begin
No
Continue with data transfer as in single master
End
E
Report and handle error

Figure 19-37. Multi-Master, Slave not enabled flow chart

19.4.14.2 Multi-Master - Slave enabled

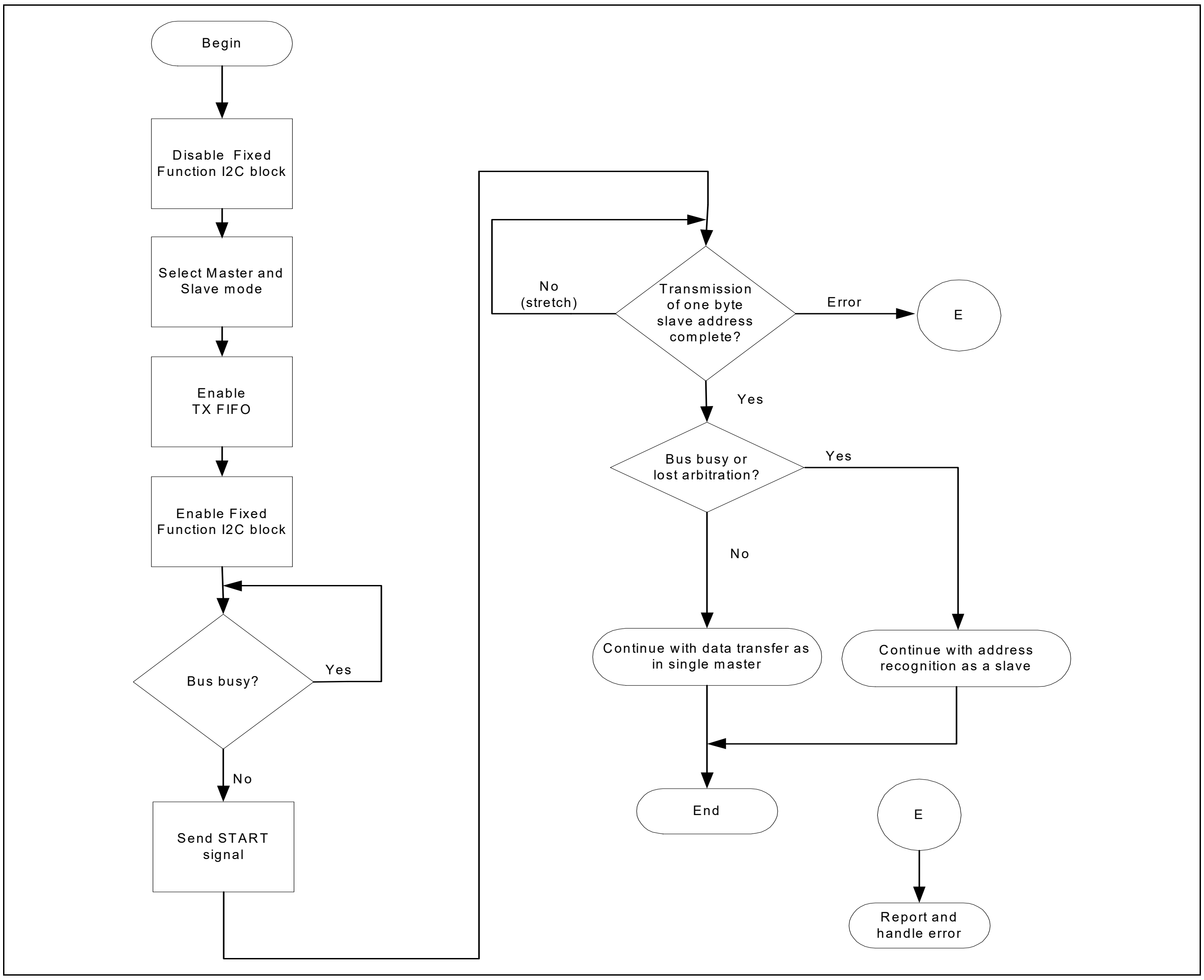

Begin
Disable Fixed Function I2C block
Select Master and Slave mode
Enable TX FIFO
Enable Fixed Function I2C block
Bus busy?
Yes
No
Send START signal
No (stretch)
Transmission of one byte slave address complete?
Error
E
Yes
Bus busy or lost arbitration?
Yes
No
Continue with data transfer as in single master
Continue with address recognition as a slave
End
E
Report and handle error

Figure 19-38. Multi-Master, Slave enabled flow chart

20 Local interconnect network (LIN)

The Local Interconnect Network (LIN) block of PSoC™ 4 HV PA supports the autonomous transfer of the LIN frame to reduce CPU processing.

20.1 Features

20.1.1 LIN

- LIN protocol support in hardware according to ISO 17987 standard
- Master and slave functionality
- Master node
  - Autonomous header transmission and autonomous response transmission and reception
- Slave node
  - Autonomous header reception and autonomous response transmission and reception
- Message buffer for PID, data, and checksum fields
- Classic and enhanced checksum
- Timeout detection
- Error detection
- Test modes including hardware error injection
- Baud rate detection
- 16x bit time oversampling

20.2 Block diagram

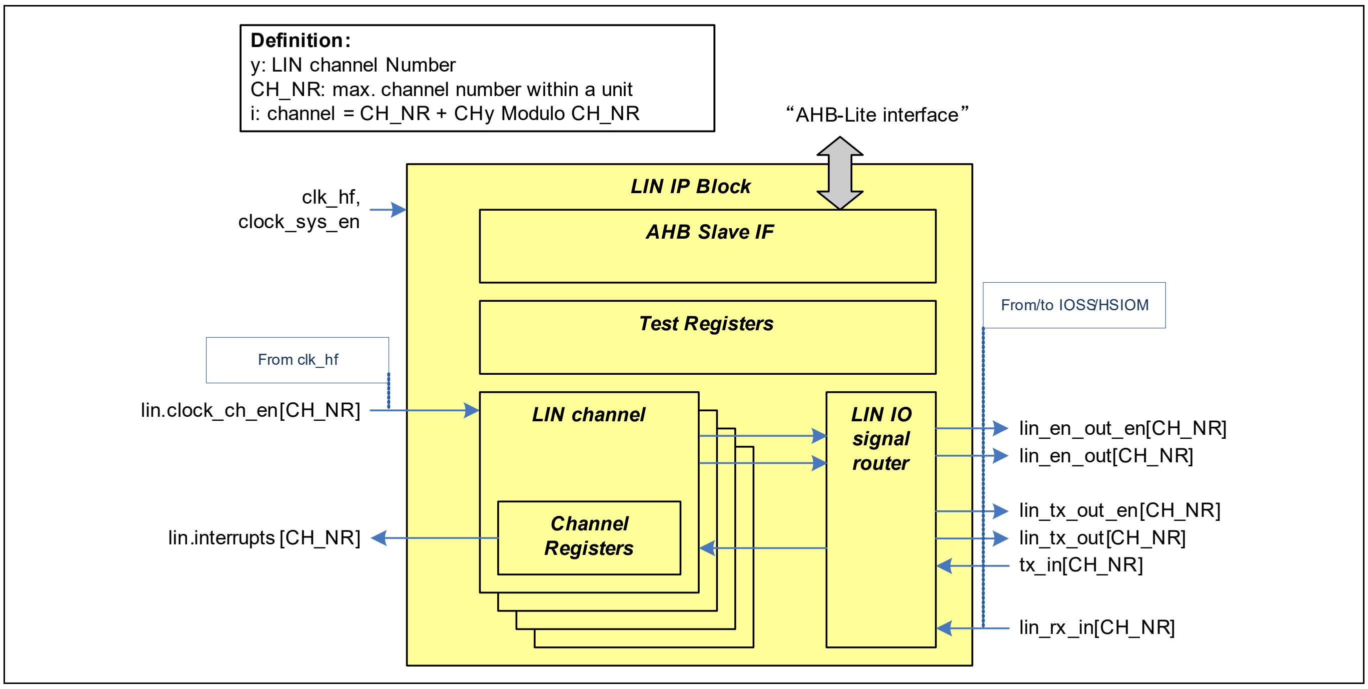


Figure 20-39. LIN block diagram

20.2.1 Internal bus interface

The LIN unit registers are connected via an AHB-Lite IF to the peripheral bus.

20.2.2 Test registers

The test error injection and different LIN signal tests are controlled within this unit.

20.2.3 LIN channel

The LIN channels are part of one common LIN unit. Each channel has its own control and status registers and its interrupts are routed to the external interrupt controller.

20.2.4 Trigger

The tr_cmd_tx_header signal is used to support "time triggered" transfers. The trigger is pulse signal more than two clock_hf cycles, which edge will be detected in LIN block. The trigger source for LIN block is selected in the PERI_TR_GROUP3_TR_OUT_CTLx[3:0] register.

Table 20-33 and **Table 20-34** provide the PSoC™ 4 HV PA LIN trigger multiplexers and the multiplexer outputs.

Table 20-33. LIN Trigger sources

PERI_TR_GROUP3_TR_OUT_CTL x[3:0]	Trigger source
0	Software trigger
1	TCPWM 0 compare match
2	TCPWM 1 compare match
3	TCPWM 2 compare match
4	TCPWM 3 compare match
5	GPIO input trigger 0
6	GPIO input trigger 1
7	GPIO input trigger 2
8	GPIO input trigger 3

Table 20-34. Trigger Multiplexer Outputs

Output	Trigger source
0	LIN timed trigger, channel #0
1	LIN timed trigger, channel #1

20.3 Clocking

Each LIN channel has its own LIN channel input clock, lin.clock_ch_en[y]. This LIN internal channel clock and register clock are derived from the high-frequency clock (clock_hf), and the peripheral clock divider settings in the clock tree, lin.clock_ch_en[y].

20.3.1 Baud rate and sample point

One LIN bit length corresponds to 16 lin.clock_ch_en[y] cycles; that is, 16 oversample counters are executed. The LIN receiver starts counting after the detection of the falling edges on the synchronized rx_synced signal to identify START bits. A bit value is sampled when the oversample counter changes from '7' to '8'.

The LIN receiver can operate (detect and sample) on the internally rx_synced signal directly, or it can operate on a filtered version of this signal by setting the LIN0_CHy_CTL0.FILTER_EN bit. The filter consists of a three-input median/majority filter that effectively performs a majority vote on a window of three consecutively rx_synced samples. For more details, see **"Noise filter"** on page 223.

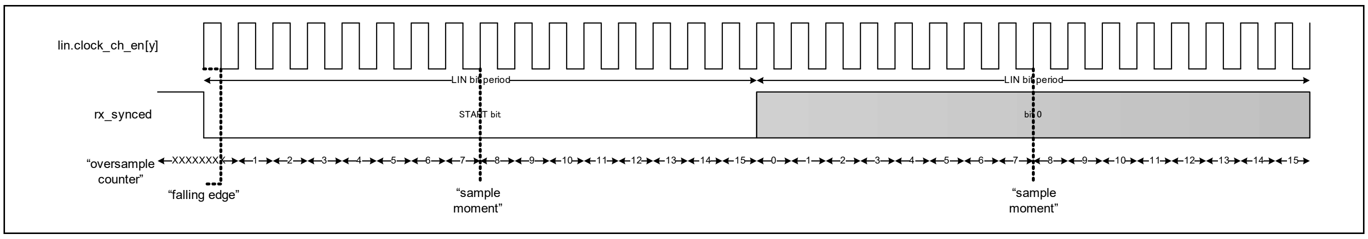


Figure 20-40. LIN Bit timing diagram

The baud rate can be configured for each channel individually, which is derived from the clock_hf. As there is the fixed signal oversampling factor of 16 in the LIN channel, for the target baud rate the clock divider for the dedicated lin.clock_ch_en[y] in the component must be calculated as follows. Thereby the baud rate calculation considered for the master resp. the slave with fixed clock and for the slave with required baud rate adjustment due to inaccurate system clock. For details about the possible the clock divider settings, see the **"Clocking system"** on page 93.

Depending on whether a fractional clock divider or an integer clock divider is applied for the LIN module input clock, check if the maximum permitted relative tolerance of the nominal LIN bit time according to the LIN ISO specification is exceeded or not. For example, the maximum master bit rate deviation from nominal bitrate (FTOL_RES_MASTER) is ± 0.5 percent.

Table 20-35. Details

CLK_HF	internal high-frequency clock
lin.clock_ch_en[y]	dedicated internal LIN channel clock derived from the internal peripheral clock
Tbit	$16 \times T_{\text{lin.clock\_ch\_en[y]}}$
f_{bit}	LIN baud rate
$f_{\text{CLK\_HF}}$	high-frequency clock frequency
CLK_DIV	clock divider for dedicated LIN channel

20.3.1.1 Baud rate calculation for LIN Master and Fixed LIN Slave Clock

$$\mathrm{CLK\_DIV} = \frac{f_{\mathrm{CLK\_SYS}}}{f_{\mathrm{LIN.CLOCK\_CH\_EN[y]}}} = \frac{f_{\mathrm{CLK\_SYS}}}{16 \times f_{\mathrm{bit}}} \tag{20.1}$$

20.3.1.2 Baud rate calculation Adjusted LIN Slave Clock

$$\mathrm{CLK\_DIV} = \frac{f_{\mathrm{CLK\_SYS}}}{f_{\mathrm{LIN.CLOCK\_CH\_EN[y]}}} \times \mathrm{SyncByteCorrection} \tag{20.2}$$

$$\mathrm{CLK\_DIV} = \frac{f_{\mathrm{CLK\_SYS}}}{16 \times f_{\mathrm{bit}}} \times \frac{\text{"LIN\_CH\_TX\_RX\_STATUS.SYNC\_COUNTER"value}}{128} \tag{20.3}$$

20.3.1.3 Example: Master

$f_{\mathrm{bit,nom}}$	nominal bit rate 20 kBaud = 20 kHz
$f_{\mathrm{bit,real}}$	real bit rate
$f_{\mathrm{CLK\_HF}}$	48 MHz

integer clock divider in use

$$\mathrm{CLK\_DIV} = \frac{f_{\mathrm{CLK\_SYS}}}{16 \times f_{\mathrm{bit,nom}}} = \frac{48\mathrm{MHz}}{16 \times 20\mathrm{kHz}} = 150 \tag{20.4}$$

As an integer clock divider is in use, the relative bit time tolerance is checked with CLK_DIV = 150.

$$f_{\mathrm{bit,real}} = \frac{f_{\mathrm{CLK\_SYS}}}{16 \times \mathrm{CLK\_DIV}} = \frac{48\mathrm{MHz}}{16 \times 150} \approx 20.00\mathrm{kHz} \tag{20.5}$$

The resulting relative bit time tolerance is 0% and within the ±0.5% of FTOL_RES_MASTER.

20.4 LIN message frame format

A LIN message frame consists of two main elements, header and response (see **Figure 20-41**).

- A frame header, transmitted only by the master node, consists of a break field, followed by a synchronization (SYNC) field and a protected identifier (PID) field.
- A frame response consisting of a maximum of eight data fields and followed by a checksum field can be transmitted by the master node or by a slave node.

With exception of the LIN break field the LIN frame structure is based on byte fields, each with a START bit and a STOP bit. Due to frame support in the LIN module registers are provided for the PID field, data fields, and checksum field. The LIN break and SYNC field are processed in the LIN module and thus there is no message buffer required for the transmission as LIN master. The handling as master or slave is controlled implicitly by commands instead of a dedicated master or slave control bit.

The following sections describe the LIN protocol support by hardware.

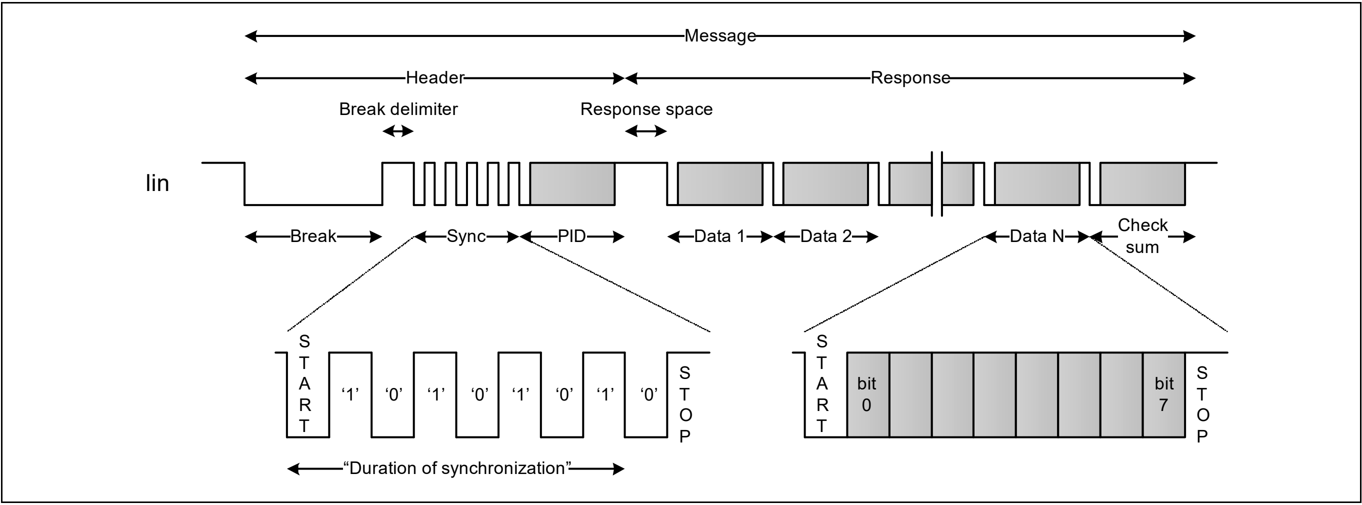


Figure 20-41. LIN Message Frame Format

20.4.1 Break and synchronization fields

The break field is generated by the master node with minimum 13-bit periods (on the master clock), whereas a slave node must detect a break field after 11-bit periods (on the slave clock). For the master and slave, the break length must be configured in LIN0_CHy_CTL0.BREAK_WAKEUP_LENGTH and the break delimiter length in LIN0_CHy_CTL0.BREAK_DELIMITER_LENGTH.

The SYNC field with the signal pattern 0x55 is used to synchronize the slave clocks to the master clock. When the LIN module is configured as master (LIN0_CHy_CMD.TX_HEADER = 1 and LIN0_CHy_CMD.RX_HEADER = 0), the SYNC field is generated autonomously. When the LIN channel is configured as slave node (LIN0_CHy_CMD.TX_HEADER = 0 and LIN0_CHy_CMD.RX_HEADER = 1), the detected baud rate is mirrored in the implicitly by the LIN0_CHy_TX_RX_STATUS.SYNC_COUNTER.

Notes:

- Before the wakeup transmission start, the bus level of the LIN signal input LIN0_CHy_TX_RX_STATUS.RX_IN must be on recessive level (logical 1). If the bus level is dominant level (logical 0), then the LIN module waits until the bus level is changing to the recessive level.
- The received signal pattern of the synchronization field is verified. When it is invalid, the error flag LIN0_CHy_INTR.RX_HEADER_SYNC_ERROR is activated.

Baud rate adjustment

The baud rate detection is done by the 128 bit lin.clock_ch_en[y] synchronization field counter (see **"Baud rate and sample point"** on page 209). The slave measures the duration of the 8-bit field, which starts from the falling edge of SYNC field START bit and stops counting with falling edge of the seventh data bit. One bit period corresponds to 16 lin.clock_ch_en[y] cycles and 8-bit periods are finally 128 lin.clock_ch_en[y] cycles.

The following table lists the synchronization cases with the resulting SYNC byte correction factor for the new clock divider calculation. The clock divider calculation for the synchronized slave is shown in **"Baud rate and sample point"** on page 209.

Table 20-36. Baud Rate Adjustment Correction Factor

Clock Ratio: master to slave	Slave Value LIN0_CHy_TX_RX_STATUS.SYNC_COUNTER	Counter Action	SYNC Byte Correction Factor for LIN ch. Clock Divider
$f_{master} = f_{slave}$	x = 128	No change	(128 / 128) = 1
$f_{master} < f_{slave}$	x > 128	Decrease the slave clock (increase the LIN ch. clock divider)	(x / 128) > 1
$f_{master} > f_{slave}$	x < 128	Increase the slave clock (decrease the LIN ch. clock divider)	(x / 128) < 1

20.4.2 PID field

The 8-bit PID field consists of a 6-bit frame identifier and a 2-bit parity over the frame identifier, for which the LIN0_CHy_PID_CHECKSUM register is provided exclusively.

- Master operation: Before the transmission start of the message frame the PID field will be written.
- Slave operation: After the reception of the STOP bit from the PID field the LIN0_CHy_PID_CHECKSUM register is updated. The confirmation of a finished and valid LIN header reception is flagged by LIN0_CHy_INTR.RX_HEADER_DONE.
- The parity of the received PID field is verified. In case of verification failure, the error flag LIN0_CHy_INTR.RX_HEADER_PARITY_ERROR is activated.

20.4.3 Data fields

As well the master as a slave can transmit a response field including maximum eight data fields. As message buffer for the data fields LIN0_CHy_DATA0 and LIN0_CHy_DATA1 are provided. The target number of data fields is processed in the register bit field LIN0_CHy_CTL1.DATA_NR. The status of transferred numbers of data bytes including the checksum field within a response is given in LIN0_CHy_STATUS.DATA_IDX. Additionally the status of an ongoing frame transfer is represented, when LIN0_CHy_STATUS.HEADER_RESPONSE is ‘1’. All these registers are used for response transmission and response reception.

The response transfer can be aborted by disabling the LIN channel (clear LIN0_CHy_CTL0.ENABLED to ‘0’).

20.4.3.1 Response transmission (LIN0_CHy_CMD.TX_RESPONSE)

Before the transmission response is started by the command LIN0_CHy_CMD.TX_RESPONSE, it must be ensured, that the data is written into the message buffer and the data length is stored.

Master operation: The response transmission can be prepared either after the reception of the PID or before the LIN frame transmission, to reduce the CPU load.

Slave operation: no additional note.

20.4.3.2 Response reception (LIN0_CHy_CMD.RX_RESPONSE)

The response reception is enabled by the command LIN0_CHy_CMD.RX_RESPONSE. It is strongly recommended, to enable it before each LIN frame start. Otherwise there is the risk of losing the response data, when the response reception is enabled after another node has already started to transmit the response. The configuration of data response length in LIN0_CHy_CTL1.DATA_NR and the checksum type selection must be configured at latest before the reception of the STOP bit in the first data byte.

Master operation: To reduce the CPU load, the data length can be stored before the LIN frame, as it is already known to the master.

Slave operation: The correct data length can be stored after the reception of the PID field. Therefore it is recommended, to configure the maximum data length for the response reception before the LIN frame transmission, to avoid timing constraints in the PID processing.

Notes: When the LIN response transmission and reception are active, both the transmission and reception error flags occur simultaneously. The transmitted data fields in the LIN0_CHy_DATA0/1 registers are not overwritten by the received data fields.

20.4.4 Checksum field

The checksum field provides an integrity check over the response data fields and optionally over the header PID field, which is controlled by the LIN0_CHy_CTL1.CHECKSUM_ENHANCED register field. The checksum field is supported through a message buffer register in LIN0_CHy_PID_CHECKSUM.CHECKSUM.

20.4.4.1 Response transmission (LIN0_CHy_CMD.TX_RESPONSE)

For the completion of the response transmission the checksum value is calculated by hardware and is transmitted automatically after the last data field. For an invalid checksum read back the LIN0_CHy_INTR.TX_RESPONSE_BIT_ERROR is set. The checksum type selection can be done already before the LIN frame start.

20.4.4.2 Response reception (LIN0_CHy_CMD.RX_RESPONSE)

When receiving, the checksum over the received PID field and data fields is calculated to verify the received checksum field. In case of verification failure a LIN0_CHy_STATUS.RX_RESPONSE_CHECKSUM_ERROR is activated. The checksum type should selected before the reception of the first data byte STOP bit reception.

20.5 Timeout operation

For development purposes a timeout functionality is provided to determine an incomplete LIN message frame operation. The timeout detection mode can be selected between a complete frame (header and response), header, and response transfer by the LIN0_CHy_CTL1.FRAME_TIMEOUT_SEL field and the timeout value is specified by the LIN0_CHy_CTL1.FRAME_TIMEOUT field in number of bit periods. The LIN0_CHy_INTR.TIMEOUT flag is set, when either the timeout detected or the stop condition is reached.

Note: An ongoing frame transfer is not aborted due to a time out.

Table 20-37. Timeout selection

FRAME_TIMEOUT_SEL Bit Field Value	Timeout Selection	Timer Start	Timer Stop
0	Timeout disabled	None	None
1	Frame mode	Falling edge of START bit in break field	Checksum field STOP bit OR timeout
2	Frame header mode	Falling edge of START bit in break field	PID field STOP bit OR timeout
3	Frame response mode	End of STOP bit	Checksum field STOP bit OR timeout

20.6 Wakeup

When a LIN cluster is in sleep state, a wakeup signal can initiate a transfer to operational state. Both the dominant wake up signal generation and detection are supported in hardware.

20.6.1 Wakeup signal transmission

Before the generation of the dominant wake up signal, its dominant pulse length should be defined in the register field LIN0_CHy_CTL0.BREAK_WAKEUP_LENGTH in bit periods, which corresponds to the specified wake up pulse length range according to the LIN specification. The transmission starts by setting LIN0_CHy_CMD.TX_WAKEUP. The flag LIN0_CHy_INTR.TX_WAKEUP_DONE confirms the completed dominant wakeup pulse, except when the received signal is different to the generated one, then the error is LIN0_CHy_INTR.TX_BIT_ERROR is set.

Note: Before the wakeup transmission start, the bus level of the LIN signal input LIN0_CHy_TX_RX_STATUS.RX_IN must be on recessive level (logical 1). If the bus level is dominant level (logical 0), then the LIN module waits until the bus level is changing to the recessive level.

20.6.2 Wakeup signal reception

To activate the wakeup reception, the commands LIN0_CHy_CMD.TX_HEADER and LIN0_CHy_CMD.RX_HEADER should be disabled.

Typically, external transceivers support remote wakeup detection. The generated 'low' level signal can be detected by polling of the receiver input LIN0_CHy_TX_RX_STATUS.RX_IN within the LIN unit. Other opportunities such as an input capture detection of the falling edge need to be checked for the dedicated port pin.

The coding information of the TX and RX transceiver pins about the wake up source can be captured directly with the internal LIN module signals LIN0_CHy_TX_RX_STATUS.RX_IN and LIN0_CHy_TX_RX_STATUS.TX_IN. For this case the LIN0_TX GPIO input function must be enabled (see the **"I/O system"** on page 141).

When the external LIN transceiver is in operational mode, the dominant wake up pulse is passed on. To detect it, the minimum expected pulse length must be configured in the form of bit periods in the register bit field LIN0_CHy_CTL0.BREAK_WAKEUP_LENGTH. When the rising edge of the dominant pulse is detected, then the flag LIN0_CHy_INTR.RX_BREAK_WAKEUP_DONE is set.

20.6.3 Wakeup in Low-Power mode

The LIN unit cannot detect a wakeup condition, when the device is Deep Sleep power mode. To support a CPU wakeup, refer to the interrupt on falling edge support for the LIN0_RX port pin of the LIN channel.

20.7 External transceiver control

Discrete LIN transceiver devices may consume a significant amount of power when enabled. Fortunately, most transceivers support the Sleep power mode in which power consumption is reduced. To this end, most transceivers have an enable "en" input signal to control the power mode.

Each LIN channel has an "en" line that is used to control the transceiver enable input signal. Before a message transfer, the en line should be activated, and after the message transfer the en line can be deactivated. The en line can be controlled by either software or hardware.

- Software control requires setting LIN0_CHy_TX_RX_STATUS.EN_OUT to '1' before a message transfer and clearing LIN0_CHy_TX_RX_STATUS.EN_OUT to '0' after a message transfer.
- Hardware control ensures setting LIN0_CHy_TX_RX_STATUS.EN_OUT to '1' before a message transfer and clearing LIN0_CHy_TX_RX_STATUS.EN_OUT to '0' after message transfer.

The LIN0_CHy_CTL0.AUTO_EN field enables the hardware control of the en signal line.

20.8 Test modes

20.8.1 Interrupt test

To test the internal interrupt signals line within the LIN module regarding functionality, an interrupt set function is provided by the LIN0_CHy_INTR_SET register.

20.8.2 Loop-back mode

A self-test circuit allows the channels to be connected to each other, to test the LIN functionality without an external transceiver or without affecting an operational LIN cluster by enabling the register bit LIN0_TEST_CTL.ENABLED. The LIN operation configuration of the two selected channels, to operate as LIN master and LIN slave, is done as usual.

Following channel loop back connections are permitted:

- Channel [0, CH_NR-2], which is identified by the LIN0_TEST_CTL.CH_IDX register field and
- the last channel [CH_NR-1].

Note: CH_NR refers to the maximum LIN channel number.

20.8.2.1 Partial disconnect mode

In this mode both channels to be tested the loop back is done via the port pins. In this case the GPIO input function of TX port pin from channel [i] must be enabled (see the **"I/O system"** on page 141).

20.8.2.2 Full disconnect mode

In this mode the LIN channels under test are routed with each other completely inside the LIN unit (see **Figure 20-44**). There is no connection to existing port pins and thereby no impact to the LIN bus.

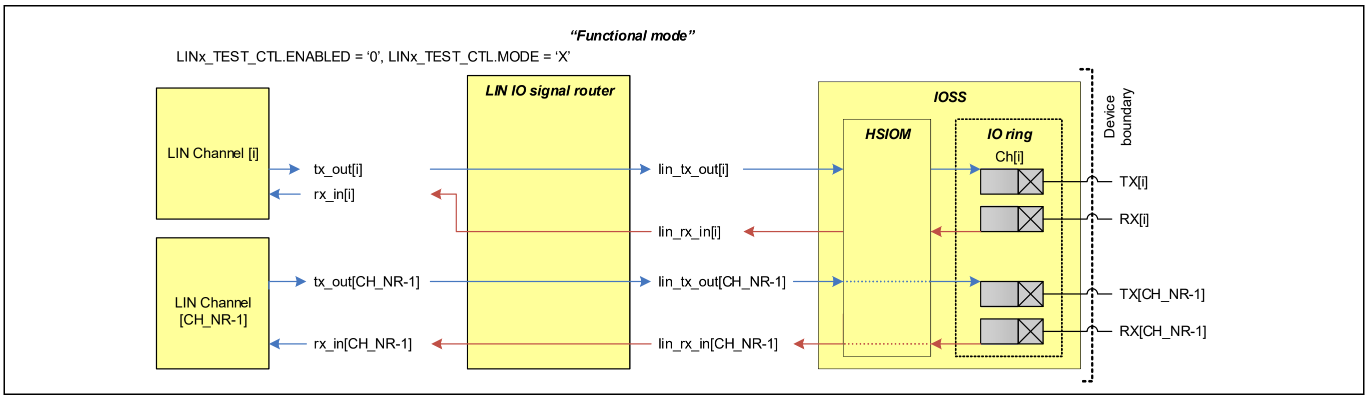


Figure 20-42. Functional Mode

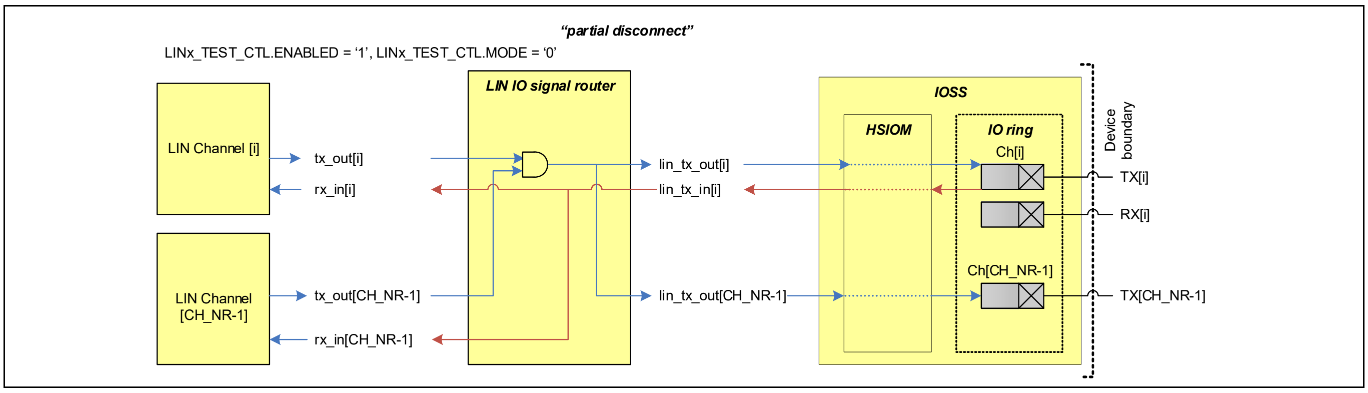


Figure 20-43. Partial Disconnect Mode

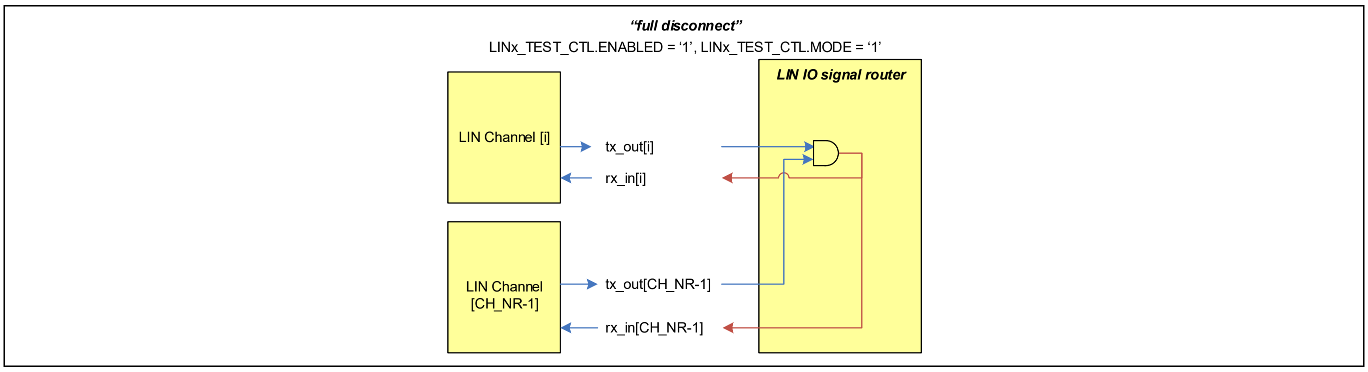


Figure 20-44. Full Disconnect Mode

20.8.3 Error injection mode

For test purposes, hardware injected transmitter errors can be generated, which result in the activation of the corresponding error flag on the reception line.

The error injection type is selected by the LIN0_ERROR_CTL register. The LIN0_ERROR_CTL.CH_IDX field specifies the channel to which the errors are applied. Table 20-38 shows the error injection types.

Table 20-38. Error Injection Support in LIN/UART Unit

Error Injection	Error Injection description	Mode support	
		LIN	UART
TX_SYNC_ERROR	The transmitted synchronization field is changed from 0x55 to 0x00.	Yes	No
TX_SYNC_STOP_ERROR	The synchronization field STOP bits are inverted to '0'.	Yes	No
TX_PARITY_ERROR	LIN: The highest parity bit of the PID field is inverted.	Yes	Yes
TX_PID_STOP_ERROR	The PID field STOP bits are inverted to '0'.	Yes	No
TX_DATA_STOP_ERROR	The data field STOP bits are inverted to '0'.	Yes	Yes
TX_CHECKSUM_ERROR	The checksum field is inverted.	Yes	No
TX_CHECKSUM_STOP_ERROR	The checksum field STOP bits are inverted to '0'.	Yes	No

20.9 Operation

20.9.1 LIN operation

20.9.1.1 LIN message transfer

The LIN protocol supports three types of message transfers:

- Master response: The master node transmits the header and transmits the response. This type can be used to control slave nodes.
- Slave response: The master node transmits the header. A slave node transmits the response and the master node receives the response. This type can be used to observe slave node status.
- Slave to slave: The master node transmits the header. A slave node transmits the response and another slave receives the response.

To support these different message types, the handling of the LIN master or LIN slave operation mode is implicitly done by command sequences.

- LIN0_CHy_CMD.TX_HEADER: This command is used exclusively by the master node to transmit a complete header such as, LIN break, SYNC field, PID field.
- LIN0_CHy_CMD.RX_HEADER: This command is used exclusively by a slave node to receive a header. After a slave node receives the header, LIN0_CHy_INTR.RX_HEADER_DONE is activated and slave node application may use the received PID field to decide to either:
  - Continue with receipt of a response (LIN0_CHy_CMD.RX_RESPONSE command).
  - Continue with transmission of a response (LIN0_CHy_CMD.TX_RESPONSE command).
  - Ignore the incoming response by disabling the channel and re-enabling for the next frame.
- LIN0_CHy_CMD.TX_RESPONSE: This command is used by the master node or a slave node to transmit a response; that is, the hardware sends the data field and the autonomously generated checksum.
- LIN0_CHy_CMD.RX_RESPONSE: This command is used by the master node or a slave node to receive a response; that is, the hardware receives the data field in one buffer and verifies the checksum.

In **Table 20-39** and **Table 20-40** the command sequences for master and slave for the different message types are shown.

Table 20-39. LIN Master Command Sequences

Message Type	Command Sequence in Register CMDi[a]			
	CMDi.TX_HEADER	CMDi.RX_HEADER	CMDi.TX_RESPONSE	CMDi.RX_RESPONSE
Master Response	1	0	1	0
Slave Response	1	0	0	1
Slave-to-Slave Response	1	0	0	0

a) Command sequence can be done before frame start.

Table 20-40. LIN Slave Command Sequences

Message Type	Command Sequence in Register CMDi[a]			
	CMDi.TX_HEADER	CMDi.RX_HEADER	CMDi.TX_RESPONSE	CMDi.RX_RESPONSE
Master Response	0	1	0	1
Slave Response	0	1	1	1[b]
Slave-to-Slave Response (transmitting node)	0	1	1	1
Slave-to-Slave Response (receiving node)	0	1	0	1
Ignore Response	0	1	0	0

a) LIN0_CHy_CMD.RX_HEADER and LIN0_CHy_CMD.RX_RESPONSE are enabled before break detection to avoid break loss and loss of data bytes in response. Disabling of LIN0_CHy_CMD.RX_RESPONSE after PID reception is permitted.

b) When both LIN0_CHy_CMD.TX_RESPONSE and LIN0_CHy_CMD.RX_RESPONSE is set, then a bus collision can be detected by LIN0_CHy_INTR.RX_RESPONSE_DONE.

Master

The master node needs to enable one interrupt cause (LIN0_CHy_INTR.TX_HEADER_DONE, LIN0_CHy_INTR.TX_RESPONSE_DONE, LIN0_CHy_INTR.RX_RESPONSE_DONE) and only enters the associated interrupt handler once.

Slave

The slave nodes will always set both LIN0_CHy_CMD.RX_HEADER and LIN0_CHy_CMD.RX_RESPONSE commands to '1'. The received header PID field will specify if a slave node must:

- Receive a response.
- Transmit a response.
- Abort the transfer and ignore the response.

By setting LIN0_CHy_CMD.RX_HEADER and LIN0_CHy_CMD.RX_RESPONSE simultaneously, the slave node anticipates response reception, to avoid loss of data bytes in the response.

Master and slave

When a message transfer is successful, the commands are cleared to '0' and must be enabled again for the next transfer. On a detected error, the transmission commands are cleared to '0', but the reception commands are not. This behavior is essential to support break-while-receive functionality on a slave node.

Both the response commands LIN0_CHy_CMD.TX_RESPONSE and LIN0_CHy_CMD.RX_RESPONSE can be enabled in parallel, a command order is processed in following priority:

- Highest priority: LIN0_CHy_CMD.TX_RESPONSE command.
- Middle priority: LIN0_CHy_CMD.RX_RESPONSE command.
- Lowest priority: No response as indicated by the absence of BOTH the LIN0_CHy_CMD.TX_RESPONSE and LIN0_CHy_CMD.RX_RESPONSE commands.

20.9.1.2 LIN software flow chart

This section shows software flow charts for the LIN master and slave operation.

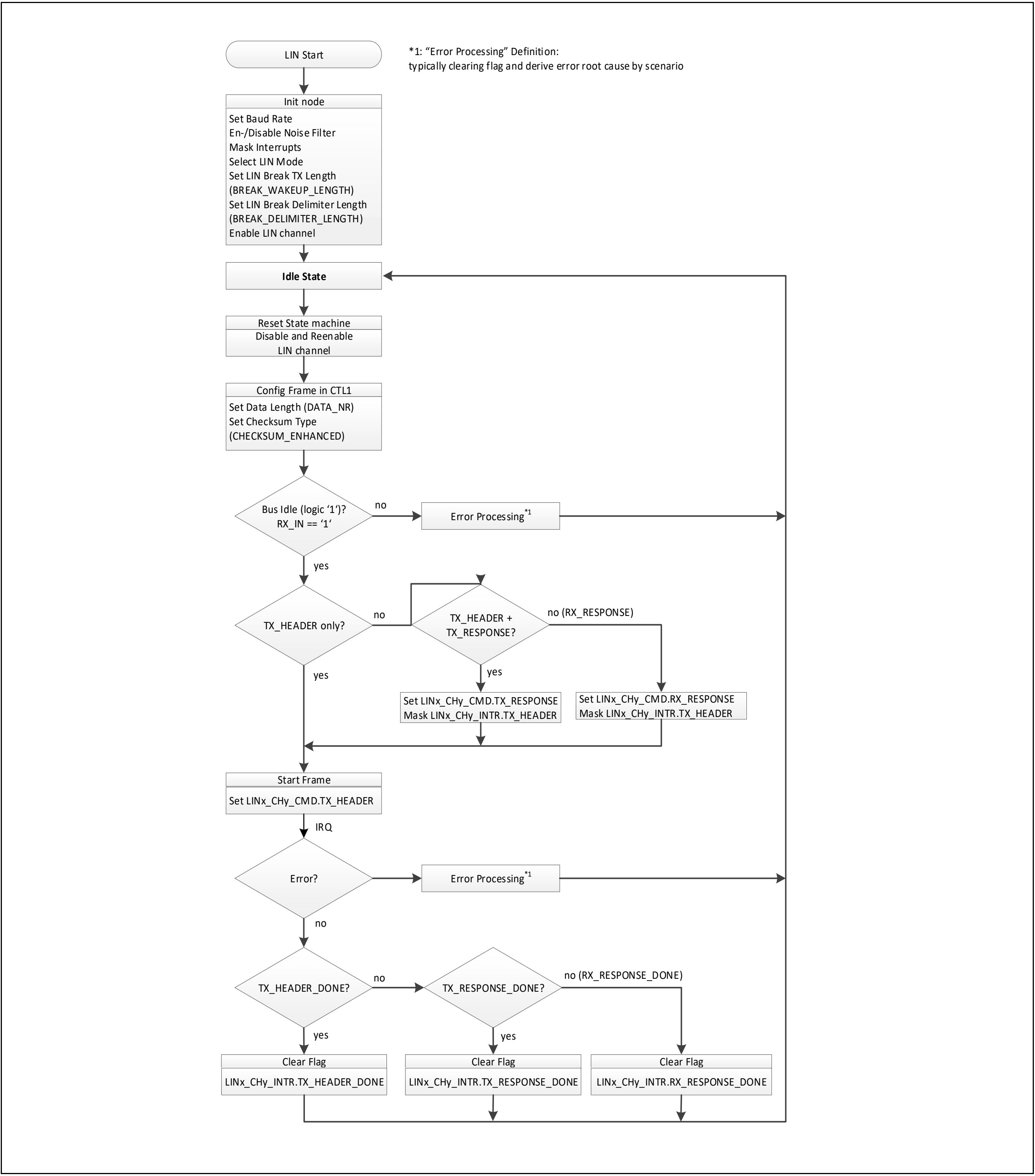


Figure 20-45. LIN Master software flow chart

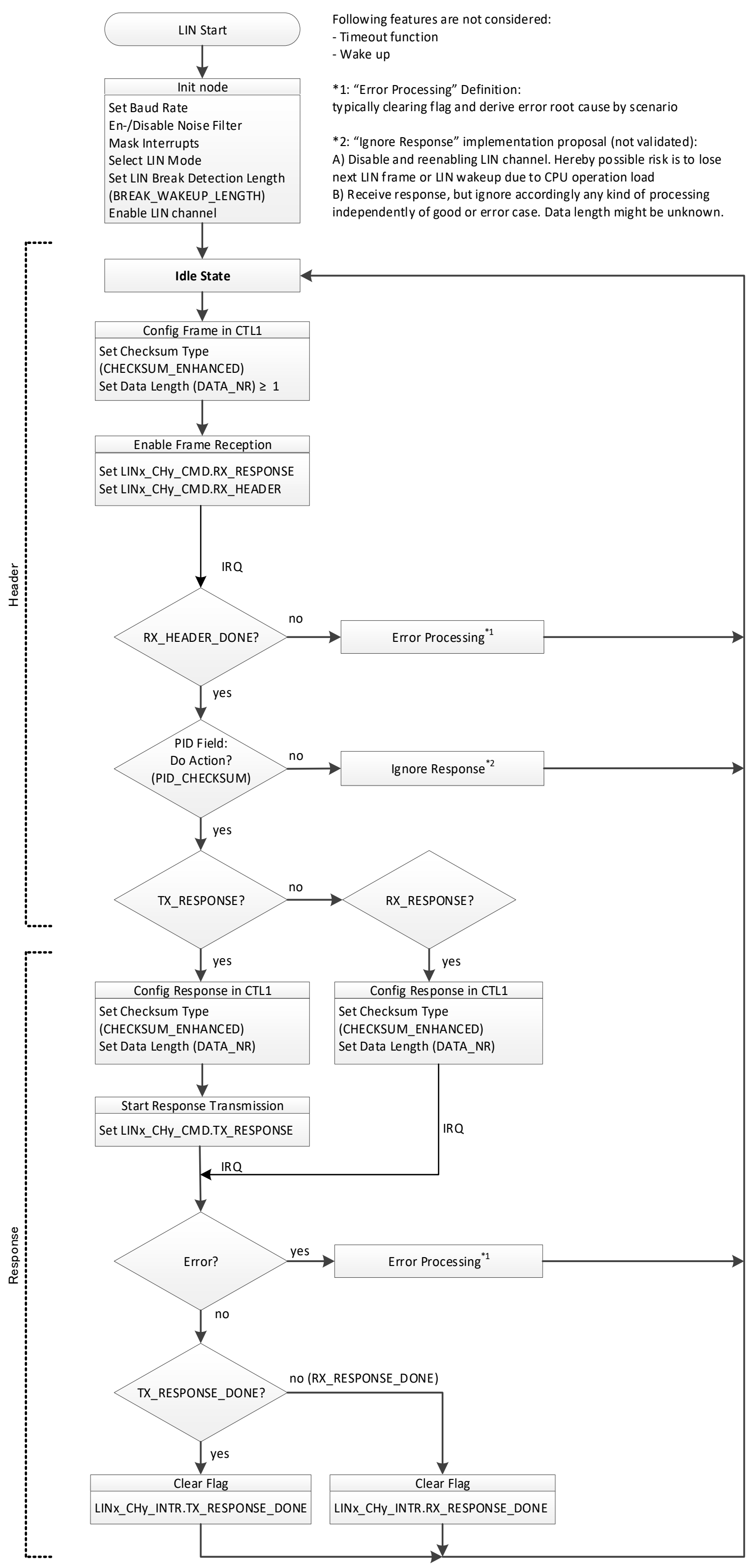

LIN Start
Init node
Set Baud Rate
En-/Disable Noise Filter
Mask Interrupts
Select LIN Mode
Set LIN Break Detection Length (BREAK_WAKEUP_LENGTH)
Enable LIN channel
Following features are not considered:
- Timeout function
- Wake up
*1: "Error Processing" Definition:
typically clearing flag and derive error root cause by scenario
*2: "Ignore Response" implementation proposal (not validated):
A) Disable and reenabling LIN channel. Hereby possible risk is to lose next LIN frame or LIN wakeup due to CPU operation load
B) Receive response, but ignore accordingly any kind of processing independently of good or error case. Data length might be unknown.
Header
Idle State
Config Frame in CTL1
Set Checksum Type (CHECKSUM_ENHANCED)
Set Data Length (DATA_NR) ≥ 1
Enable Frame Reception
Set LINx_CHy_CMD.RX_RESPONSE
Set LINx_CHy_CMD.RX_HEADER
IRQ
RX_HEADER_DONE?
no
Error Processing*1
yes
PID Field: Do Action? (PID_CHECKSUM)
no
Ignore Response*2
yes
TX_RESPONSE?
no
RX_RESPONSE?
yes
yes
Response
Config Response in CTL1
Set Checksum Type (CHECKSUM_ENHANCED)
Set Data Length (DATA_NR)
Config Response in CTL1
Set Checksum Type (CHECKSUM_ENHANCED)
Set Data Length (DATA_NR)
Start Response Transmission
Set LINx_CHy_CMD.TX_RESPONSE
IRQ
IRQ
Error?
yes
Error Processing*1
no
TX_RESPONSE_DONE?
no (RX_RESPONSE_DONE)
yes
Clear Flag
LINx_CHy_INTR.TX_RESPONSE_DONE
Clear Flag
LINx_CHy_INTR.RX_RESPONSE_DONE

Figure 20-46. LIN Slave Software Flow Chart

20.10 Noise filter

The LIN receiver operates on the synchronized rx_synced input signal, as shown in **Figure 20-47**.

- When LIN0_CHy_CTL0.FILTER_EN is '0', the receiver operates on rx_synced directly.
- When LIN0_CHy_CTL0.FILTER_EN is '1', the receiver operates on the majority of the last three rx_synced signal values based on the internal module clock LIN0_lin.clock_ch_en[y]. This filter suppresses noise on the rx_in input. Note that the filter adds a delay of one cycle to the receiver. **Figure 20-47** shows the block diagram of the noise filter and **Figure 20-48** shows the noise filtering timing behavior including the sample point position.

20.10.1 Example

When a 0, 1, 0 sequence is synchronized, the '1' is filtered out due to majority decision for '0'.

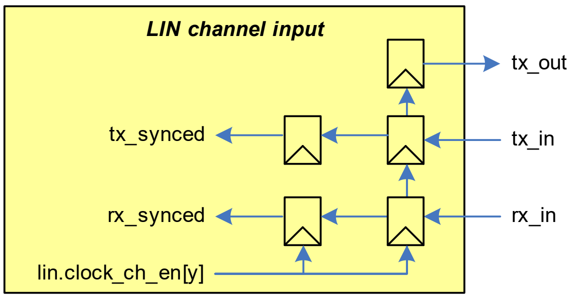


Figure 20-47. LIN signal line synchronization block diagram

Even when the median filter effectively eliminates the rx_in noise, it is of interest to be notified of this noise, as the noise can be an indication of a malfunctioning LIN cluster. Therefore, the receiver verifies the rx_in signal by investigating the last three rx_synced signal values, which are the same values as used by the median filter. The verification consists of two types:

- Sampling verification
  When a START bit, a data bit or STOP bit value is sampled (in the middle of a bit period), all three rx_synced signal values should be the same (a 0, 0, 0 sequence or a 1, 1, 1, sequence).
- Generic verification
  The isolated '0' or '1' values may not occur (a 1, 0, 1 sequence or a 0, 1, 0 sequence)

When the noise filter is enabled (LIN0_CHy_CTL0.FILTER_EN is ‘1’), the error flag LIN0_CHy_INTR.RX_NOISE_DETECT is set in case of a verification failure. An ongoing frame is not aborted by the noise detection.

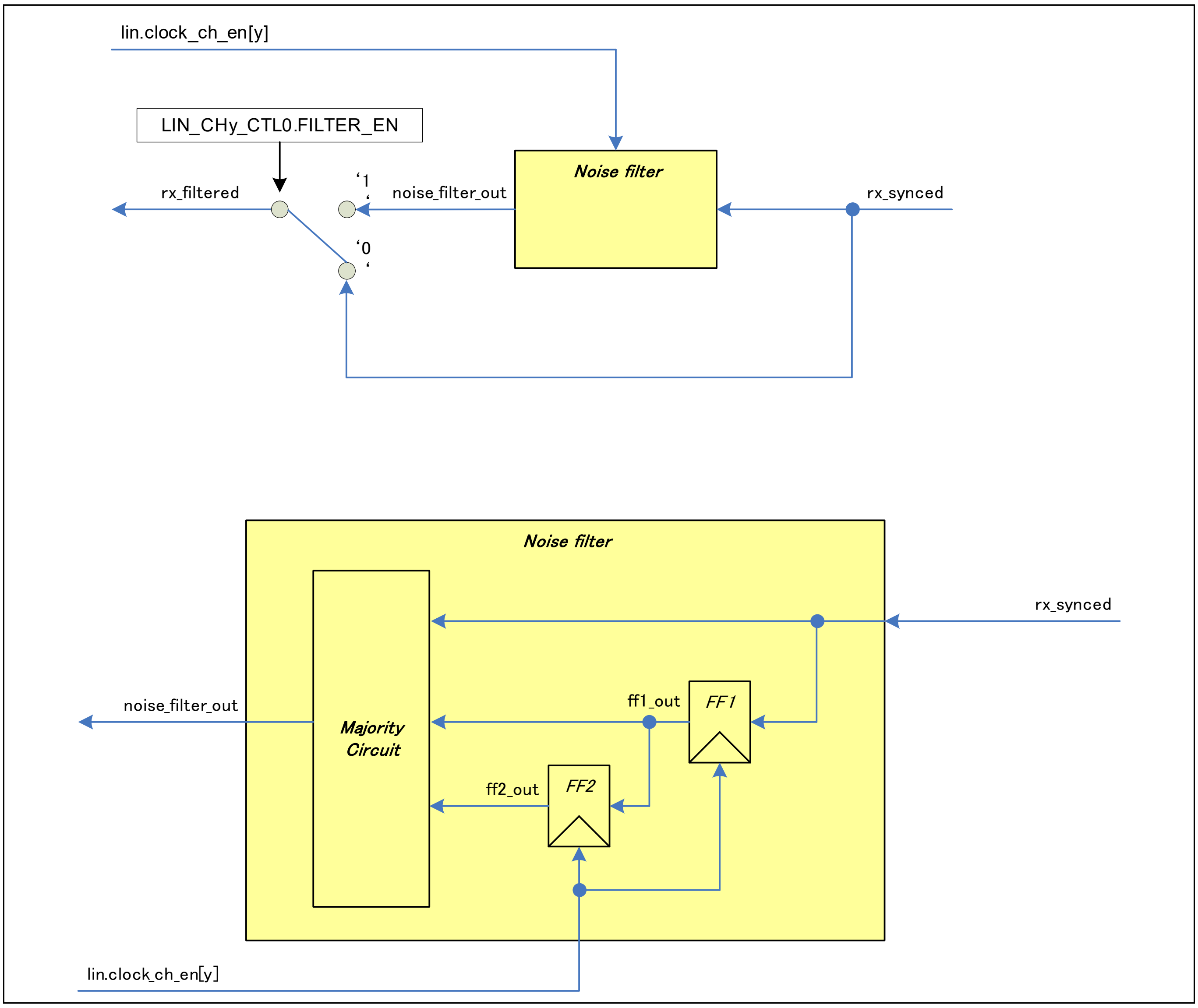


Figure 20-48. LIN noise filter block diagram

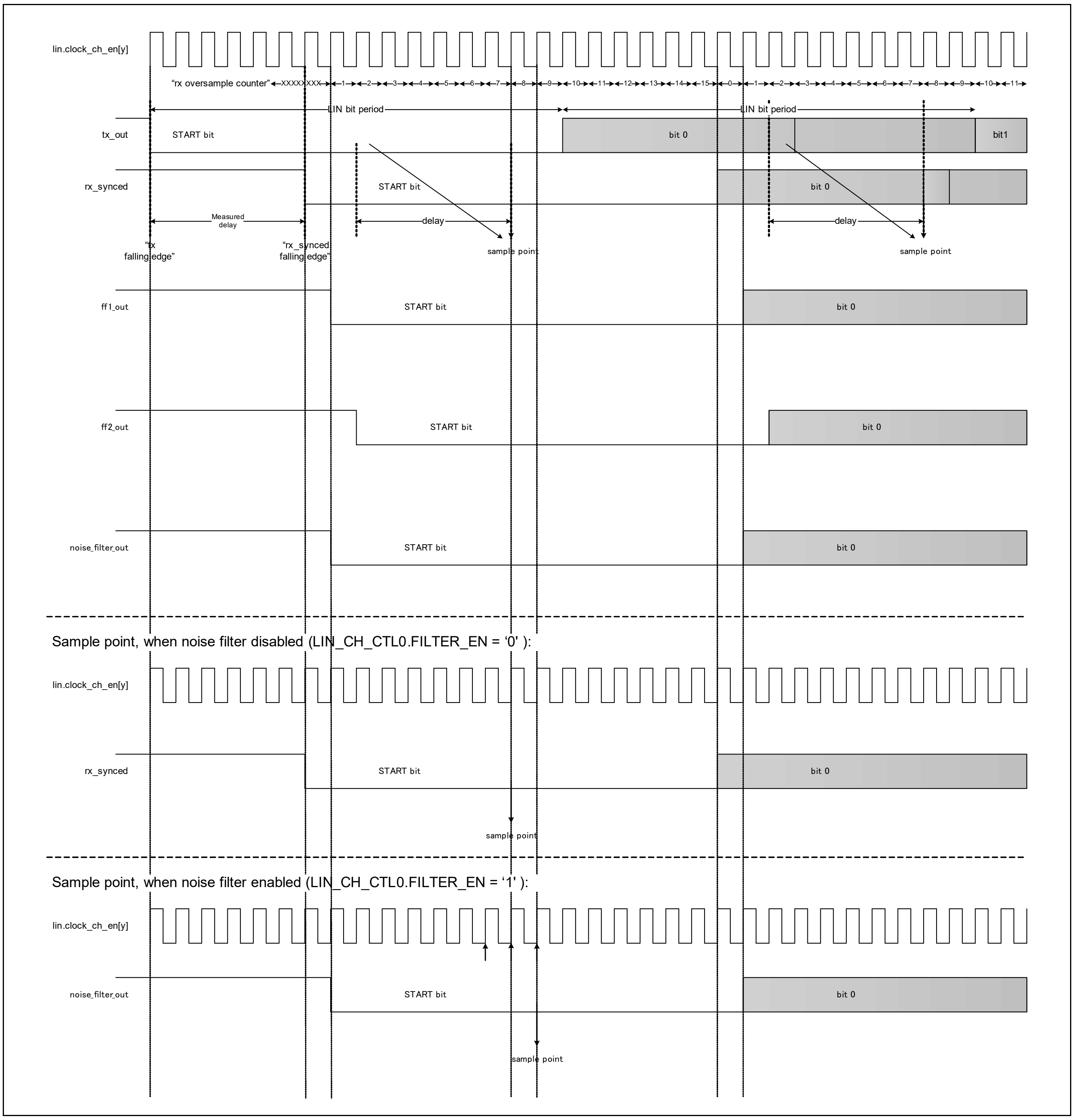


Figure 20-49. LIN noise filtering timing diagram

20.11 Interrupts and flags

20.11.1 Overview

The LIN module supports multiple LIN channels; each LIN channel has its dedicated interrupt line and accordingly its own set of interrupt registers LIN0_CHy_INTR, LIN0_CHy_INTR_SET, LIN0_CHy_INTR_MASK, and LIN0_CHy_INTR_MASKED.

To reduce interrupt load of the interrupt source flags listed in the LIN0_CHy_INTR register, an AND masking is done with the LIN0_CHy_INTR_MASK. The masked interrupts, which cause interrupt on the interrupt controller, are shown in the LIN0_CHy_INTR_MASKED register.

Table 20-41. Details

	Data	Register
	00000111	LIN0_CH_INTR
AND	00000111	LIN0_CH_INTR_MASK
	00000111	LIN0_CH_INTR_MASKED

The following tables give an overview of the interrupt events in the module in different modes.

Table 20-42. Interrupt Events in LIN Master Mode

Event Type	Event	Event Detection Condition	Clear Event Flag	Transfer Abort	Enable Interrupt	Register Flag Bit
TX	Header Transmission done	Header transmission succeeded	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	–	yes	LIN0_CHy_INTR. TX_HEADER_DONE
TX	Response Transmission done	Response transmission succeeded	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	–	yes	LIN0_CHy_INTR. TX_RESPONSE_DONE
TX	Wakeup Transmission done	Wake up signal successfully transmitted	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	–	yes	LIN0_CHy_INTR. TX_WAKEUP_DONE
RX	Response Reception done	Response reception succeeded	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	–	yes	LIN0_CHy_INTR. RX_RESPONSE_DONE
RX	Wakeup Reception done	Wake up signal received, after wake up reception detection was enabled.	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	–	yes	LIN0_CHy_INTR. RX_BREAK_WAKEUP_ DONE
Error	Time out	A frame, header or response does not finish within a specified time	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	no	yes	LIN0_CHy_INTR. TIMEOUT

Table 20-42. Interrupt Events in LIN Master Mode (continued)

Event Type	Event	Event Detection Condition	Clear Event Flag	Transfer Abort	Enable Interrupt	Register Flag Bit
TX Error	Transmitter Header Bit Error	The incoming bus level does not match with the transmitted value during: • header transmission • wake up transmission	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	yes[a)]	yes	LIN0_CHy_INTR. TX_HEADER_BIT_ ERROR.
TX Error	Transmitter Response Bit Error	During the response transmission the received bus value does not match with the transmitted value	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	yes[a]	yes	LIN0_CHy_INTR. TX_RESPONSE_BIT_ ERROR
RX Error	Noise Detection	Noise on RX input detected, when LIN0_CHy_CTL0. FILTER_EN is '1'	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	no	yes	LIN0_CHy_INTR. RX_NOISE_DETECT
RX Error	Receiver Response Frame Error	An invalid start bit or stop bit occurs during response reception (data field, checksum)	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	yes	yes	LIN0_CHy_INTR. RX_RESPONSE_ FRAME_ ERROR
RX Error	Receiver Response Checksum Error	The calculated checksum over the data bytes and optionally the PID field does match with the received checksum.	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	yes	yes	LIN0_CHy_INTR. RX_RESPONSE_ CHECKSUM_ERROR

a) When LIN0_CHy_CTL0.BIT_ERROR_IGNORE is '1', then bit errors are still reported, but do not abort an ongoing transfer.

Table 20-43. Interrupt Events in LIN Slave Mode

Event Type	Event	Event Detection Condition	Clear Event Flag	Transfer Abort	Enable Interrupt	Register Flag Bit
TX	Response Transmission done	Response transmission succeeded	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	–	yes	LIN0_CHy_INTR. TX_RESPONSE_DONE
TX	Wakeup Transmission done	Wake up signal successfully transmitted	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	–	yes	LIN0_CHy_INTR. TX_WAKEUP_DONE
RX	Header Reception done	Header reception succeeded	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	–	yes	LIN0_CHy_INTR. RX_HEADER_DONE
RX	Response Reception done	Response reception succeeded	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	–	yes	LIN0_CHy_INTR. RX_RESPONSE_DONE
RX	Wakeup Reception done	Wake up signal received, after wake up reception detection was enabled.	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	–	yes	LIN0_CHy_INTR. RX_BREAK_WAKEUP_ DONE
RX	Synchronization Field Reception done	Synchronizati on field successfully received	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	–	yes	LIN0_CHy_INTR. RX_HEADER_SYNC_ DONE
Error	Time out	A frame, header or response does not finish within a specified time	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	no	yes	LIN0_CHy_INTR. TIMEOUT
TX Error	Transmitter Response Bit Error	The incoming bus level does not match with the transmitted value during the response	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	yes[a)]	yes	LIN0_CHy_INTR. TX_RESPONSE_BIT_ ERROR

Table 20-43. Interrupt Events in LIN Slave Mode (continued)

Event Type	Event	Event Detection Condition	Clear Event Flag	Transfer Abort	Enable Interrupt	Register Flag Bit
RX Error	Noise Detection	noise on RX input detected, when LIN0_CHy_ CTL0. FILTER_EN is '1'	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	no	yes	LIN0_CHy_INTR. RX_NOISE_DETECT
RX Error	Receiver Header Frame Error	• An invalid start bit occurs during PID field. • An invalid stop bit occurs during SYNC or PID field.	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	yes	yes	LIN0_CHy_INTR. RX_HEADER_FRAME_ ERROR
RX Error	Receiver Synchronization Error	An invalid data field pattern is detected during the reception of the SYNC field	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	yes	yes	LIN0_CHy_INTR. RX_HEADER_SYNC_ ERROR
RX Error	Receiver PID Parity Error	The received PID field has a parity error	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	yes	yes	LIN0_CHy_INTR. RX_HEADER_PARITY_ ERROR

Table 20-43. Interrupt Events in LIN Slave Mode (continued)

Event Type	Event	Event Detection Condition	Clear Event Flag	Transfer Abort	Enable Interrupt	Register Flag Bit
RX Error	Receiver Response Frame Error	An invalid stop bit occurs during response reception (data field, checksum)	• Write ‘1’ to flag • LIN0_CHy_CTL. ENABLED to ‘0’	yes	yes	LIN0_CHy_INTR. RX_RESPONSE_ FRAME_ERROR
RX Error	Receiver Response Checksum Error	The calculated checksum over the data bytes and optionally the PID field does match with the received checksum.	• Write ‘1’ to flag • LIN0_CHy_CTL. ENABLED to ‘0’	yes	yes	LIN0_CHy_INTR. RX_RESPONSE_ CHECKSUM_ERROR

a) When LIN0_CHy_CTL0.BIT_ERROR_IGNORE is ‘1’, then bit errors are still reported, but do not abort an ongoing transfer.

Table 20-44. Interrupt Events in UART Mode

Event Type	Event	Event Detection Condition	Clear Event Flag	Transfer Abort	Enable Interrupt	Register Flag Bit
TX	Transmission done	Transmission succeeded	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	-	yes	LIN0_CHy_INTR. TX_HEADER_DONE
RX	Reception done	Reception succeeded	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	-	yes	LIN0_CHy_INTR. RX_HEADER_DONE
TX Error	Transmitter Bit Error	The incoming bus level does not match with the transmitted value during transmission.	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	yes[a)]	yes	LIN0_CHy_INTR. TX_HEADER_BIT_ERRO R
RX Error	Noise Detection	noise on RX input detected, when LIN0_CHy_CTL0. FILTER_EN is '1'.	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	no	yes	LIN0_CHy_INTR. RX_NOISE_DETECT
RX Error	Receiver Frame Error	An invalid start bit resp. stop bit occurs during reception.	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	yes	yes	LIN0_CHy_INTR. RX_HEADER_FRAME_ ERROR
RX Error	Receiver Parity Error	The received PID field has a parity error.	• Write '1' to flag • LIN0_CHy_CTL. ENABLED to '0'	yes	yes	LIN0_CHy_INTR. RX_HEADER_PARITY_ ERROR

a) When LIN0_CHy_CTL0.BIT_ERROR_IGNORE is '1', then bit errors are still reported, but do not abort an ongoing transfer.

20.11.2 Transmission

20.11.2.1 TX header done

After a successful header transmission as master the flag LIN0_CHy_INTR.TX_HEADER_DONE is activated. This means, the flag is set after the valid PID STOP bit verification. The enabled command bits such as LIN0_CHy_CMD.TX_HEADER within this frame session are not cleared, as long as a selected legal command sequence (see **"LIN operation"** on page 218) is not successfully completed.

Clearing the flag

The flag can be cleared either by a write access to the flag with '1' within the LIN0_CHy_INTR register or disabling the LIN channel (LIN0_CHy_CTL0.ENABLED = 0).

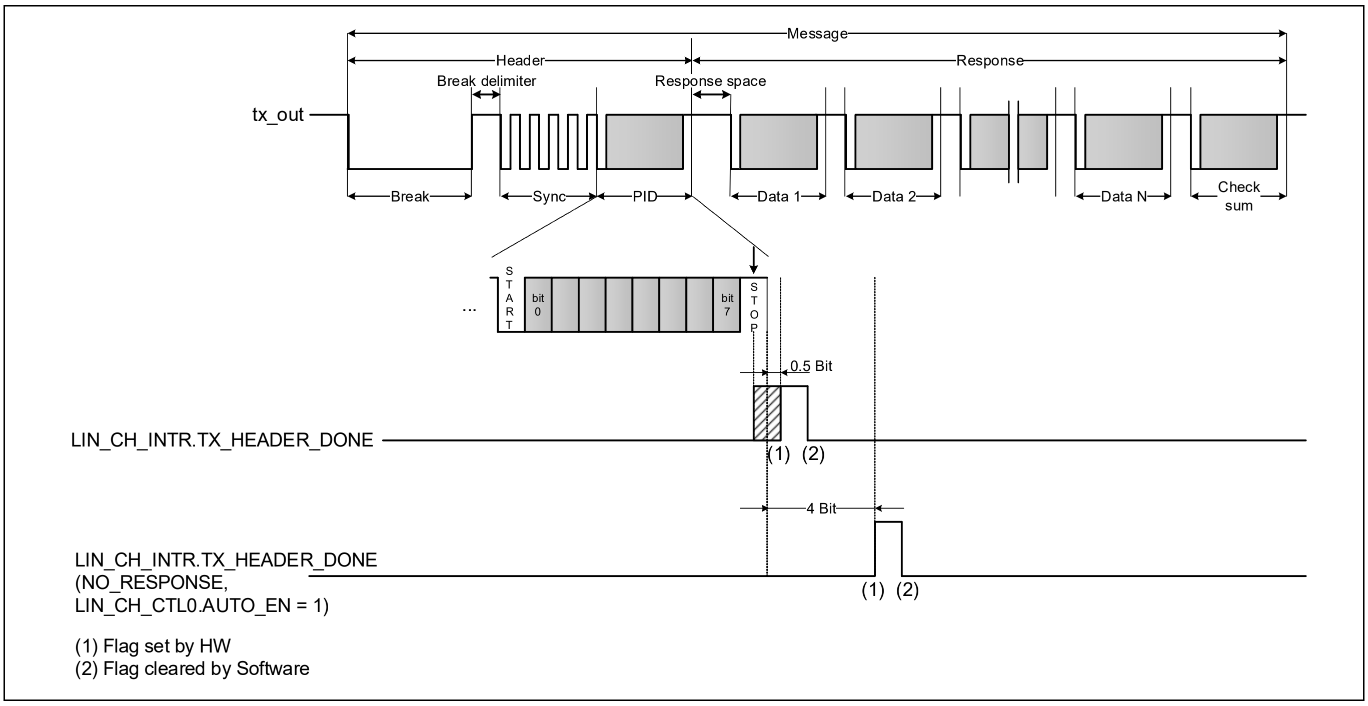


Figure 20-50. TX header done flag timing diagram

20.11.2.2 TX response done

After a valid completion of a frame including the CHECKSUM STOP bit, the LIN0_CHy_INTR.TX_RESPONSE_DONE flag is activated; that is, the flag is set after the valid CHECKSUM STOP bit verification. The enabled commands such as LIN0_CHy_CMD.TX_RESPONSE within this frame session are not cleared, as long as a selected legal command sequence (see **"LIN operation"** on page 218) is not successfully completed.

Clearing the flag

The flag can be cleared either by a write access to the flag with '1' within the LIN0_CHy_INTR register or disabling the LIN channel (LIN0_CHy_CTL0.ENABLED = 0).

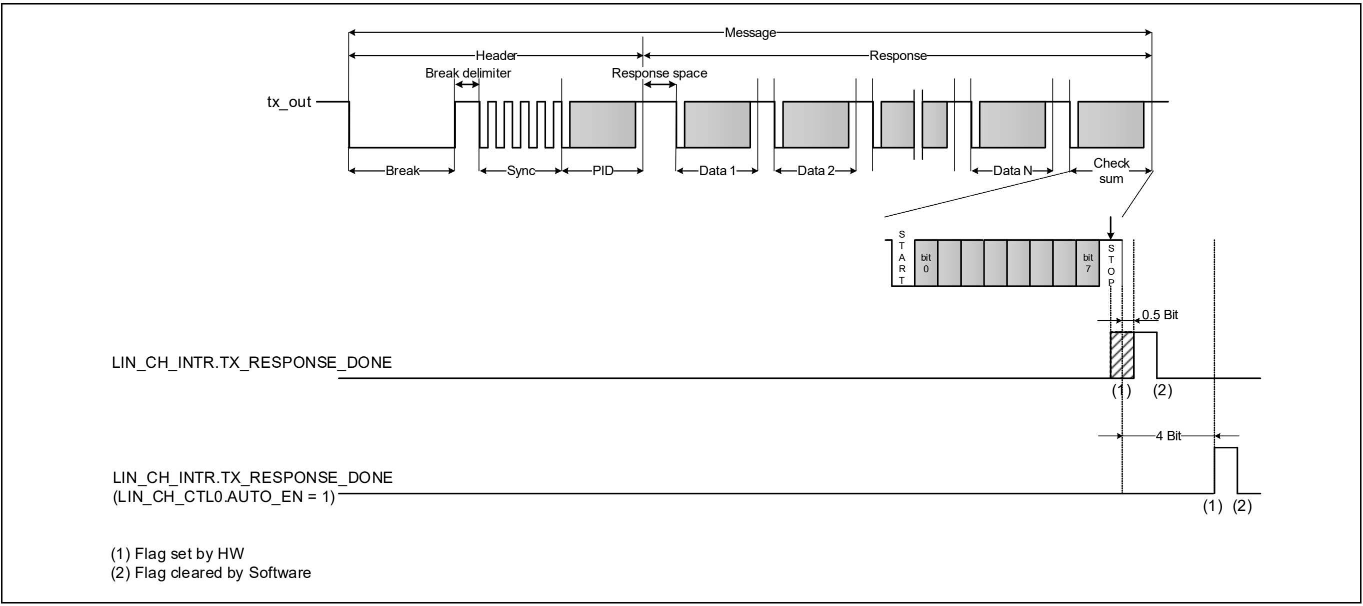


Figure 20-51. TX response done flag timing diagram

20.11.2.3 TX Wakeup done

To support remote wakeup detection, the header reception commands LIN0_CHy_CMD.RX_HEADER and LIN0_CHy_CMD.TX_HEADER are in cleared state. At the end of the successfully transmitted dominant wake up pulse the flag LIN0_CHy_INTR.TX_WAKEUP_DONE is set to '1'.

Clearing the flag

The flag can be cleared either by a write access to the flag with '1' within the LIN0_CHy_INTR register or disabling the LIN channel (LIN0_CHy_CTL0.ENABLED = 0).

Note: The flag is not set when LIN0_CHy_INTR.TX_HEADER_BIT_ERROR is set due to transmission error.

20.11.3 Reception

20.11.3.1 RX break wakeup done

After transition from the break low pulse to the break delimiter bit, a break detection interrupt is set by the LIN0_CHy_INTR.BREAK_WAKEUP_DONE flag. This interrupt flag does not need to be enabled for the regular header processing.

As the wakeup function is shared with the break function the end of the wakeup pulse detection is represented by the same flag.

Clearing the flag

The flag can be cleared either by a write access to the flag with '1' within the LIN0_CHy_INTR register or disabling the LIN channel (LIN0_CHy_CTL0.ENABLED = 0).

20.11.3.2 RX Header SYNC Done

After reception of a valid SYNC byte pattern and valid SYNC STOP bit the LIN0_CHy_INTR.RX_HEADER_DONE flag is set.

Clearing the flag

The flag can be cleared either by a write access to the flag with '1' within the LIN0_CHy_INTR register or disabling the LIN channel (LIN0_CHy_CTL0.ENABLED = 0).

20.11.3.3 RX header done

After reception of a valid LIN header including a valid PID STOP bit and PID parity check, the LIN0_CHy_INTR.RX_HEADER_DONE flag is set. The command bit LIN0_CHy_CMD.RX_HEADER is not cleared, as long as a legal command sequence (see **"LIN operation"** on page 218) is not successfully completed.

Clearing the flag

The flag can be cleared either by a write access to the flag with '1' within the LIN0_CHy_INTR register or disabling the LIN channel (LIN0_CHy_CTL0.ENABLED = 0).

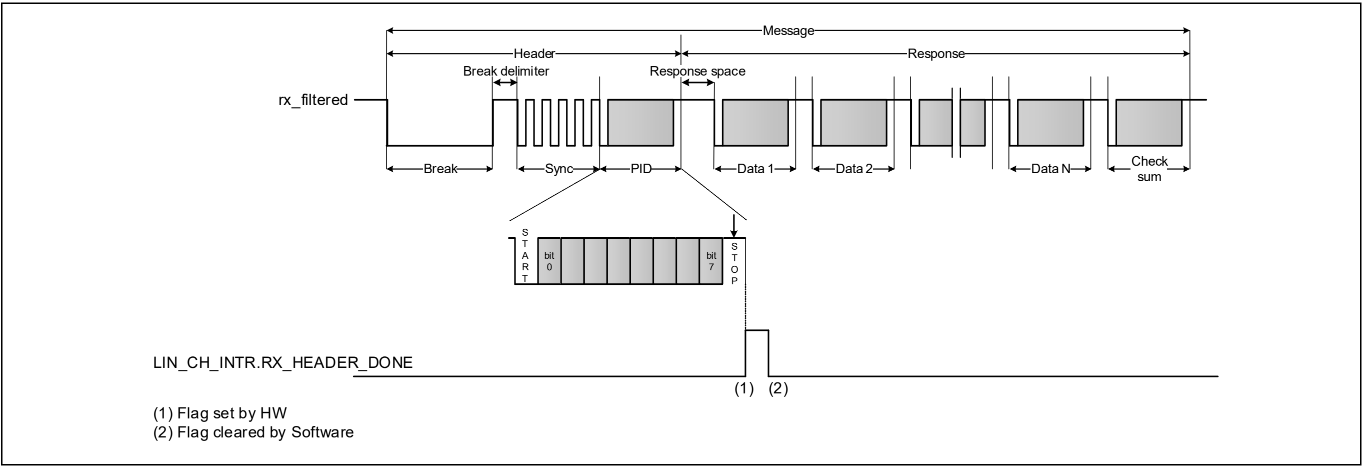


Figure 20-52. "RX Header Done" flag timing diagram

20.11.3.4 RX response done

After a valid completion of a frame including CHECKSUM STOP bit and the checksum verification the LIN0_CHy_INTR.RX_RESPONSE_DONE flag is set. The enabled commands such as LIN0_CHy_CMD.TX_RESPONSE within this frame session are not cleared, as long as a selected legal command sequence (see **"LIN operation"** on page 218) is not successfully completed.

Clearing the flag

The flag can be cleared either by a write access to the flag with '1' within the LIN0_CHy_INTR register or disabling the LIN channel (LIN0_CHy_CTL0.ENABLED = 0).

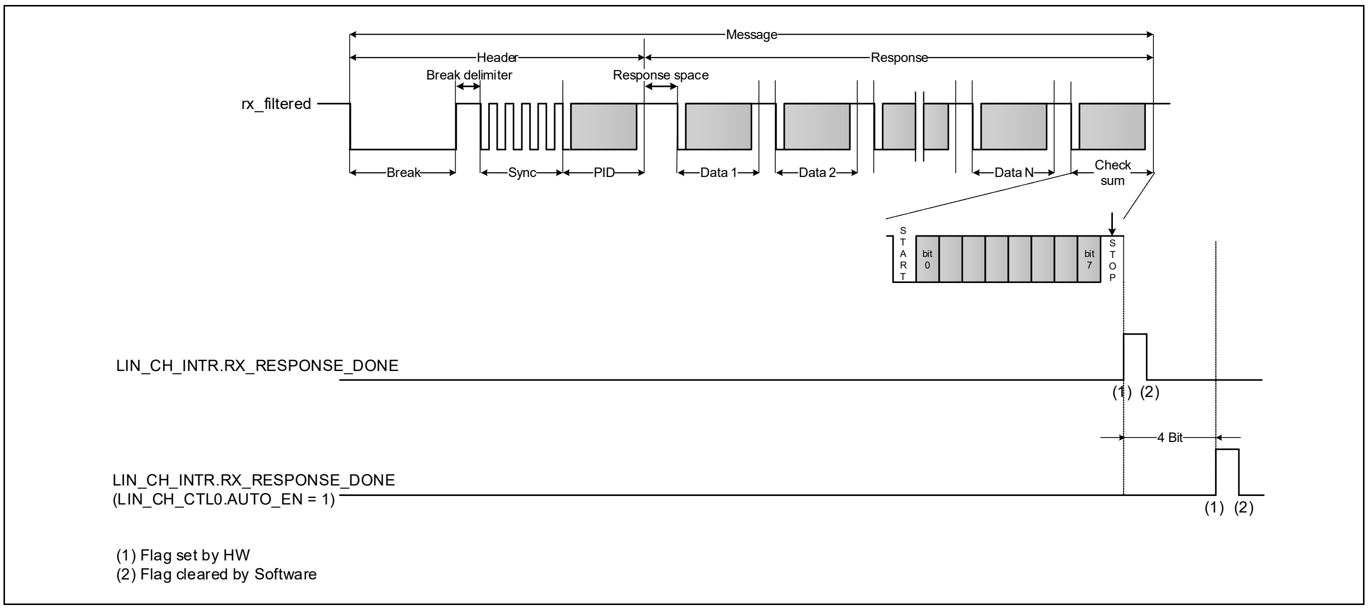


Figure 20-53. "RX Response Done" flag timing diagram

20.11.4 Error and Status detection

To ensure robust behavior, several types of errors are detected. When an error is detected, the associated interrupt cause in the INTR register is activated. **Figure 20-54** and **Figure 20-55** give an overview about the appearance of error events for the LIN master and LIN slave.

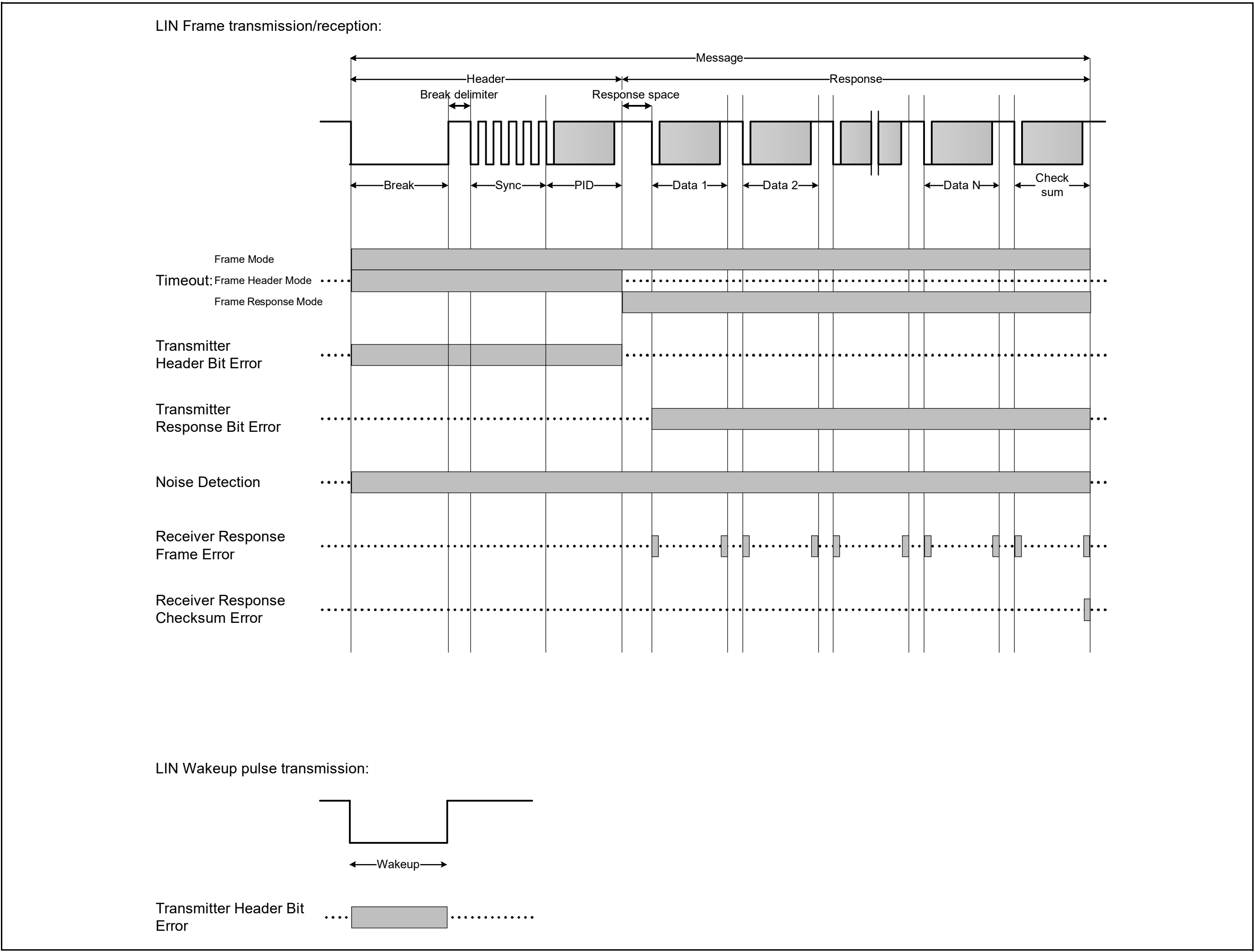

LIN Frame transmission/reception:
Message
Header
Response
Break delimiter
Response space
Break
Sync
PID
Data 1
Data 2
Data N
Check sum
Frame Mode
Timeout:
Frame Header Mode
Frame Response Mode
Transmitter Header Bit Error
Transmitter Response Bit Error
Noise Detection
Receiver Response Frame Error
Receiver Response Checksum Error
LIN Wakeup pulse transmission:
Wakeup
Transmitter Header Bit Error

Figure 20-54. LIN master error events timing diagram

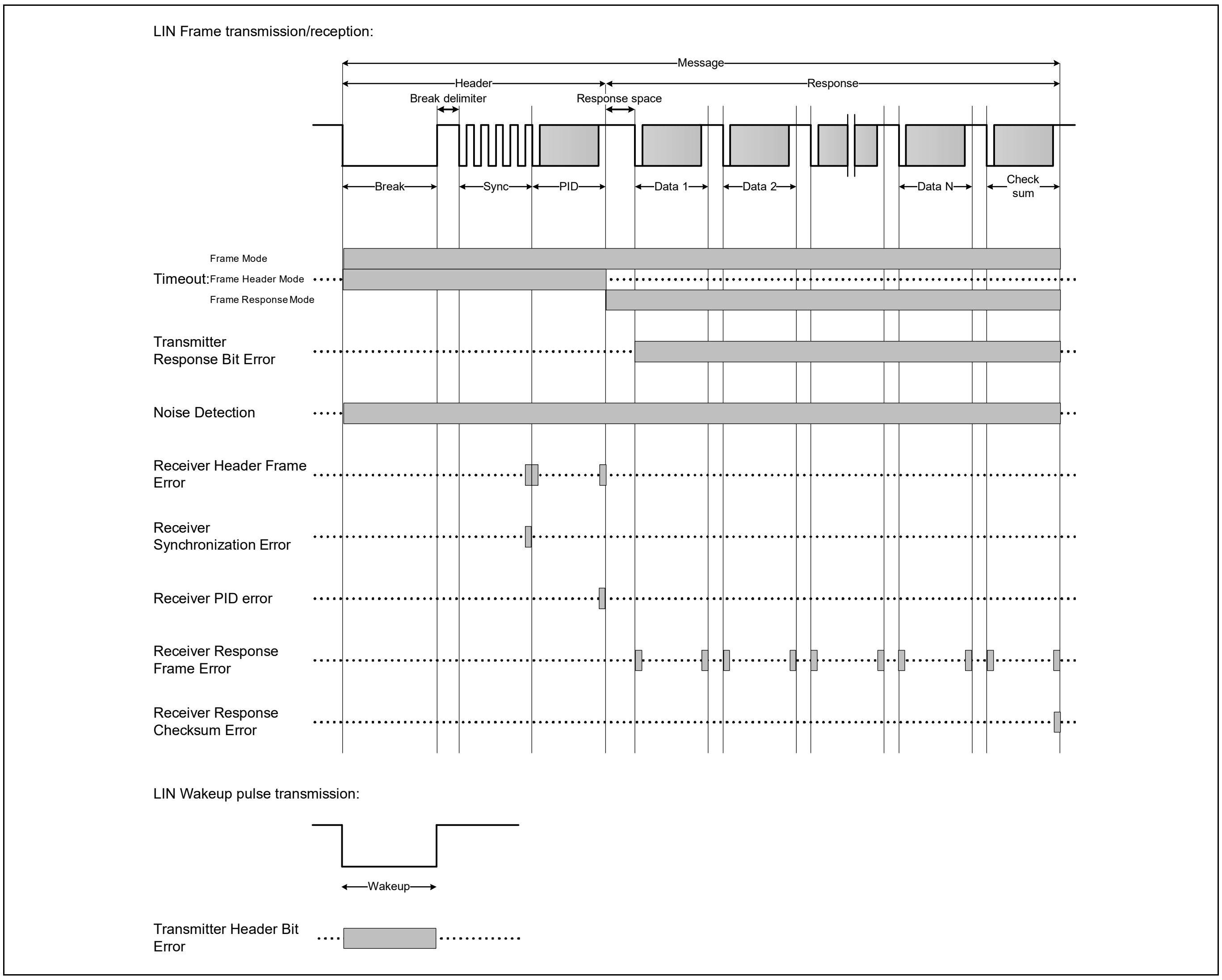

LIN Frame transmission/reception:
Message
Header
Response
Break delimiter
Response space
Break
Sync
PID
Data 1
Data 2
Data N
Check sum
Frame Mode
Timeout: Frame Header Mode
Frame Response Mode
Transmitter Response Bit Error
Noise Detection
Receiver Header Frame Error
Receiver Synchronization Error
Receiver PID error
Receiver Response Frame Error
Receiver Response Checksum Error
LIN Wakeup pulse transmission:
Wakeup
Transmitter Header Bit Error

Figure 20-55. LIN slave error events timing diagram

Notes:

- When the LIN0_CHy_CTL0.BIT_ERROR_IGNORE is '1', a bit error (the timeout error is not included) does not abort an ongoing transfer, although the bit errors are always reported.
- As the transmission commands (such as TX_REPONSE) have higher priority than the reception commands (such as RX_RESPONSE) in the processing order the transmission errors are only reported, when both commands are activated.

20.11.4.1 Transmitter bit error

During transmission the transmitted value on the RX line is also received over the TX line. The transmitted and received values should be the same. If this verification detects a failure, an LIN0_CHy_INTR.TX_HEADER_BIT_ERROR or LIN0_CHy_INTR.TX_RESPONSE_BIT_ERROR is activated and the transmission is automatically aborted by the hardware. This also includes the detection of an invalid START and STOP bit.

The error flag LIN0_CHy_INTR.TX_HEADER_BIT_ERROR is valid for:

- Break field
- Synchronization field
- PID field
- Wake up low pulse

The error flag LIN0_CHy_INTR.TX_RESPONSE_BIT_ERROR is valid for:

- Data fields
- Checksum field

Clearing the flag

Both flags can be cleared either by a write access to the flag with ‘1’ within the LIN0_CHy_INTR register or disabling the LIN channel (LIN0_CHy_CTL0.ENABLED = 0).

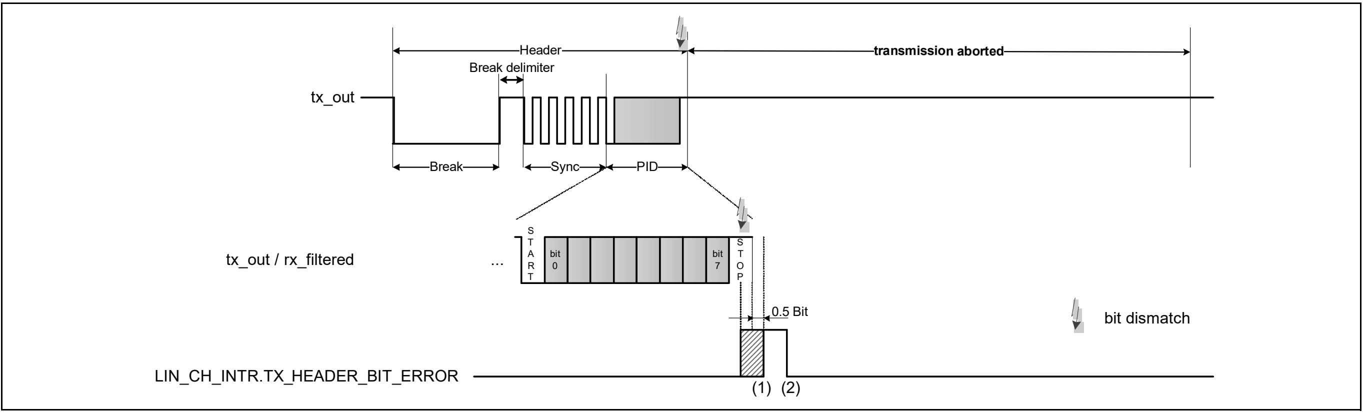


Figure 20-56. Transmitter bit error timing diagram

20.11.4.2 Receive synchronization error

The slave experiences a synchronization error, when SYNC byte pattern is either incorrect or the synchronization range is exceeded. The error is shown by the LIN0_CHy_INTR. RX_HEADER_SYNC_ERROR flag.

Clearing the flag

The flag can be cleared either by a write access to the flag with ‘1’ within the LIN0_CHy_INTR register or disabling the LIN channel (LIN0_CHy_CTL0.ENABLED = 0).

20.11.4.3 Receiver frame error

A START bit should be received as a ‘0’ on the RX line and a STOP bit should be received as a ‘1’ on the RX line. A START bit occurs at specific moments in the frame after a falling edge on the RX line and a STOP bits occurs after every 8-bit field. The error is detected after the sample time of the RX line, which is in the center of bit period (see **“Baud rate and sample point”** on page 209).

Header Reception

When a frame error is detected during the header the LIN0_CHy_INTR.RX_HEADER_FRAME_ERROR flag is set. The ongoing transfer is aborted automatically.

Response Reception

During the response, the LIN0_CHy_INTR.RX_RESPONSE_FRAME_ERROR flag is activated when the frame error occurs in the data bytes 2 to 8 or in the checksum. Additionally, the ongoing response reception is aborted by the hardware.

Exception: Framing Error in Data Byte 1

Case A: “no response”:

Here the response part is missing and followed by a LIN break of the next LIN frame. The event flag LIN0_CHy_INTR.RX_RESPONSE_DONE and error flag LIN0_CHy_INTR.RX_RESPONSE_FRAME_ERROR stays ‘0’, but the LIN0_CHy_STATUS.RX_DATA0_FRAME_ERROR is set. But the flag is only set in case of slave operation, indicated by RX_HEADER command. The response reception is not aborted by the invalid STOP bit in the data byte 1. But missing bus activity within the frame can be also detected by LIN0_CHy_INTR.TIMEOUT.

Case B: “error response”:

As in this previous case, a detected invalid STOP bit in the data byte 1 is flagged by LIN0_CHy_STATUS.RX_DATA0_FRAME_ERROR and response reception, first of all, continue with the START bit of the next byte (either data byte 2 or the checksum field). Consequently, a frame error is detected, and LIN0_CHy_INTR.RX_RESPONSE_FRAME_ERROR is set to ‘1’. Hereby the next byte is only transmitted when the frame error in data byte 1 is not detected by the transmitting node.

Note: LIN0_CHy_STATUS.RX_DATA0_FRAME_ERROR does not trigger any interrupt. It must be checked explicitly.

Clearing the flag

Both flags can be cleared either by a write access to the flag with ‘1’ within the LIN0_CHy_INTR register or disabling the LIN channel (LIN0_CHy_CTL0.ENABLED = 0). The LIN0_CHy_STATUS.RX_DATA0_FRAME_ERROR clears automatically. At the falling edge of the SYNC start bit, which means after the INTR.RX_HEADER_BREAK_WAKEUP_DONE flag.

20.11.4.4 Receiver PID parity error

The receiver calculates the parity bits over the received frame identifier in the PID field. The calculated parity bits are verified against the received parity bits in the PID field. In case of verification failure, the LIN0_CHy_INTR.RX_HEADER_PARITY_ERROR flag is set.

Clearing the flag

The flag can be cleared either by a write access to the flag with ‘1’ within the LIN0_CHy_INTR register or disabling the LIN channel (LIN0_CHy_CTL0.ENABLED = 0).

20.11.4.5 Response checksum error

The receiver calculates the checksum over the received PID field (optionally as specified by the LIN0_CHy_CTL0.CHECKSUM_ENHANCED register field) and the received data fields. The calculated checksum is verified against the received checksum field. In case of verification failure, the LIN0_CHy_INTR.RX_RESPONSE_CHECKSUM_ERROR is activated.

Clearing the flag

The flag can be cleared either by a write access to the flag with '1' within the LIN0_CHy_INTR register or disabling the LIN channel (LIN0_CHy_CTL0.ENABLED = 0).

20.11.4.6 Receiver noise detection

When the noise filter is enabled (LIN0_CHy_CTL0.FILTER_EN is '1'), the error flag LIN0_CHy_INTR.RX_NOISE_DETECT is set in case of a verification failure. But a going transfer is not aborted. See **"Noise filter"** on page 223 for more details.

Clearing the flag

The flag can be cleared either by a write access to the flag with '1' within the LIN0_CHy_INTR register or disabling the LIN channel (LIN0_CHy_CTL0.ENABLED = 0).

Note: An ongoing frame is not aborted by the noise detection.

20.11.4.7 Timeout detection

As described in **"Timeout operation"** on page 214, the timer operation inside the LIN module is supported. When one of the selected timeouts is detected, the LIN0_CHy_INTR.TIMEOUT flag is activated.

Note: The timeout detection does not abort an ongoing frame.

20.11.5 Dedicated operation use case(s)

20.11.5.1 LIN Slave Node response reception

For a slave node, it is required to distinguish a “no response” from an “error response” scenario, in addition to the typical “correct response” scenario. When LIN0_CHy_CMD.RX_HEADER and LIN0_CHy_CMD.RX_RESPONSE are set to ‘1’, a slave node expects to receive a response with a specific number of data fields. Despite this expectation, there is a possibility that the response is NOT transmitted.

- The master node may have decided to abort the frame transfer after it transmitted the header.
- Another slave node may not be operational and therefore not be able to transmit the response.

In both cases, the slave node expects to receive a response, but there is no response. Consider the following cases:

a) While waiting for the first data field of the response, there is no other bus activity. This case can be detected using the timeout functionality (LIN0_CHy_INTR.TIMEOUT).

b) While waiting for the first data field of the response, the master node transmits the header of the next frame.

If the slave node expects a data field and receives a break field of the header of the next frame, the data field’s STOP bit has a frame error.

- If a response is transmitted, the frame error is applicable and indicates an “error response”.
- If a response is not transmitted (a header of the next frame is transmitted), the frame error is not applicable and indicates a “no response”.

20.11.5.2 Different Slave Node response reception scenarios

Correct response:

A LIN0_CHy_INTR.RX_HEADER_DONE activation is followed by a LIN0_CHy_INTR.RX_RESPONSE_DONE activation and LIN0_CHy_STATUS.RX_DATA0_FRAME_ERROR is ‘0’ within the same frame.

Error response – Data Field 1:

Within the same frame, a LIN0_CHy_INTR.RX_HEADER_DONE activation is followed by a LIN0_CHy_INTR.RX_RESPONSE_DONE activation and LIN0_CHy_STATUS.RX_DATA0_FRAME_ERROR is ‘1’ because the response reception is not aborted because of a frame error in data field 1. The condition is that if the transmitting node does not detect the frame error, it aborts the transmission.

Error response – All except Data Field 1:

Within the same frame, a LIN0_CHy_INTR.RX_HEADER_DONE activation is followed by a LIN0_CHy_INTR.RX_RESPONSE_FRAME_ERROR activation, because the response reception is aborted due to frame error in the complete response field, except in data field 1. Therefore, there can be also a frame error in the first data field, shown by LIN0_CHy_STATUS.RX_DATA0_FRAME_ERROR is ‘1’.

No response:

Because of the missing response in the first frame, the LIN0_CHy_INTR.RX_HEADER_DONE activation is followed by the second frame and its header by the LIN0_CHy_INTR.RX_HEADER_DONE activation. Therefore, there is no LIN0_CHy_INTR.RX_RESPONSE_DONE activation.

20.12 Registers

Table 20-45. LIN Global Unit Registers

Register	Name	Description
LIN0_ERROR_CTL	Error Control Register	Error injection control for the full LIN unit.
LIN0_TEST_CTL	Test Control Register	Test control is done for all channels.

Table 20-46. LIN Channel Registers

Register	Name	Description
LIN0_CHy_CTL0	Control 0 Register	In this register the channel can be enabled. Furthermore the communication mode selection and mode configurations are provided.
LIN0_CHy_CTL1	Control 1 Register	Beside the LIN data length and the checksum the timeout counter is processed in the register.
LIN0_CHy_STATUS	Status Register	The communication state flags and the error flags, which are mirrored from the INTR register, are listed.
LIN0_CHy_CMD	Command Register	The communication protocol is controlled.
LIN0_CHy_TX_RX_STATUS	TX/RX Status Register	An input and output status of the LIN transceiver control is reported. Additionally the LIN synchronization counter provides a counter value, which needs to be processed for the synchronization procedure in software.
LIN0_CHy_PID_CHECKSUM	PID Checksum Register	PID and checksum buffer.
LIN0_CHy_DATA0	Data 0 Register	The response buffer for the data byte fields 0 to 3 is covered.
LIN0_CHy_DATA1	Data 1 Register	The response buffer for the data byte fields 4 to 7 is covered.
LIN0_CHy_INTR	Interrupt Register	The status of communication and error flags is shown.
LIN0_CHy_INTR_SET	Interrupt Set Register	Communication and error flags in the INTR register can be set for test purposes.
LIN0_CHy_INTR_MASK	Interrupt Mask Register	A bit mask over the communication and error flags can be defined.
LIN0_CHy_INTR_MASKED	Interrupt Masked Register	Masked communication and error flags are listed.

Note: In LIN0_CHy, 'y' is the channel number under the LIN instance.

Table 20-47. Peripheral Interconnect trigger group control registers

Register	Name	Description
PERI_TR_CTL	Trigger Control Register	This register provides software control over trigger activation.
PERI_TR_GROUP3_TR_OUT_CTLx	Trigger Control Register	This register specifies the input trigger for a specific output trigger in trigger group3.

21 Timer, Counter, and PWM

The Timer, Counter, and Pulse Width Modulator (TCPWM) block in PSoC™ 4 HV PA implements the 16-bit timer, counter, pulse width modulator (PWM), and quadrature decoder functionality. The block can be used to measure the period and pulse width of an input signal (timer), find the number of times a particular event occurs (counter), generate PWM signals, or decode quadrature signals. This chapter explains the features, implementation, and operational modes of the TCPWM block.

21.1 Features

- Up to four 16-bit timers, counters, or pulse width modulators (PWM)
- The TCPWM block supports the following operational modes:
  - Timer
  - Capture
  - Quadrature decoding
  - Pulse width modulation
  - Pseudo-random PWM
  - PWM with dead time
- Multiple counting modes – up, down, and up/down
- Clock prescaling (division by 1, 2, 4, ... 64, 128)
- Double buffering of compare/capture and period values
- Supports interrupt on:
  - Terminal Count – The final value in the counter register is reached
  - Capture/Compare – The count is captured to the capture/compare register or the counter value equals the compare value
- Complementary line output for PWMs

21.2 Block diagram

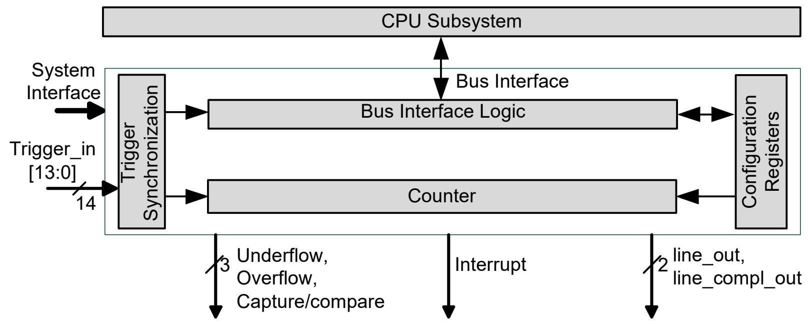


Figure 21-57. TCPWM block diagram

The block has these interfaces:

- Bus interface: Connects the block to the CPU subsystem.
- I/O signal interface: Connects input triggers (such as reload, start, stop, count, and capture) to dedicated GPIOs.
- Interrupts: Provides interrupt request signals from the counter, based on terminal count (TC) or CC conditions.
- System interface: Consists of control signals such as clock and reset from the system resources subsystem.

This TCPWM block can be configured by writing to the TCPWM registers. See **"TCPWM registers"** on page 273 for more information on all registers required for this block.

21.2.1 Enabling and disabling counter in TCPWM block

The counter can be enabled by setting the COUNTER_ENABLED field (bit 0) of the control register TCPWM_CTRL.

Note: The counter must be configured before enabling it. If the counter is enabled after being configured, registers are updated with the new configuration values. Disabling the counter retains the values in the registers until it is enabled again (or reconfigured).

21.2.2 Clocking

The TCPWM receives the high-frequency clock through the system interface to synchronize all events in the block. The counter enable signal (counter_en), which is generated when the counter is enabled, gates the high-frequency clock to provide a counter-specific clock (counter_clock). Output triggers (explained later in this chapter) are also synchronized with the high-frequency clock.

Clock Prescaling: counter_clock can be prescaled, with divider values of 1, 2, 4… 64, 128. This prescaling is done by modifying the GENERIC field of the counter control (TCPWM_CNT_CTRL) register, as shown in **Table 21-48**.

Table 21-48. Bit-field setting to prescale counter clock

GENERIC[10:8]	Description
0	Divide by 1
1	Divide by 2
2	Divide by 4
3	Divide by 8
4	Divide by 16
5	Divide by 32
6	Divide by 64
7	Divide by 128

Note: Clock prescaling cannot be done in quadrature mode and PWM-DT mode.

21.2.3 Events based on trigger inputs

These are the events triggered by hardware or software.

- Reload
- Start
- Stop
- Count
- Capture/switch

Hardware triggers can be level signal, rising edge, falling edge, or both edges. **Figure 21-58** shows the selection of edge detection type for any event trigger signal.

Any edge (rising, falling, or both) or level (high) can be selected for the occurrence of an event by configuring the trigger control register 1 (TCPWM_CNT_TR_CTRL1). This edge/level configuration can be selected for each trigger event separately. Alternatively, firmware can generate an event by writing to the counter command register (TCPWM_CMD), as shown in **Figure 21-58**.

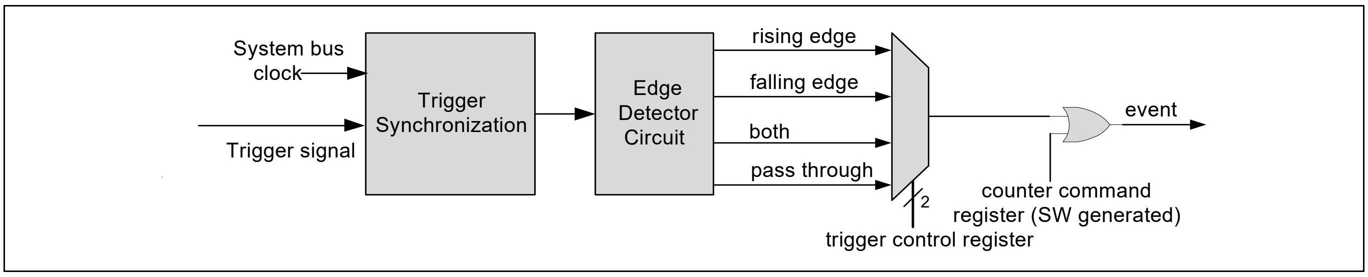


Figure 21-58. Trigger signal edge detection

The trigger signal to generate an event can be a GPIO signal, TCPWM's underflow, compare match or overflow signal, or Sample_Done signal. **Figure 21-59** shows the trigger signal selection for all the events. The trigger source for TCPWM is selected in the PERI_TR_GROUP1_TR_OUT_CTLx[5:0] register.

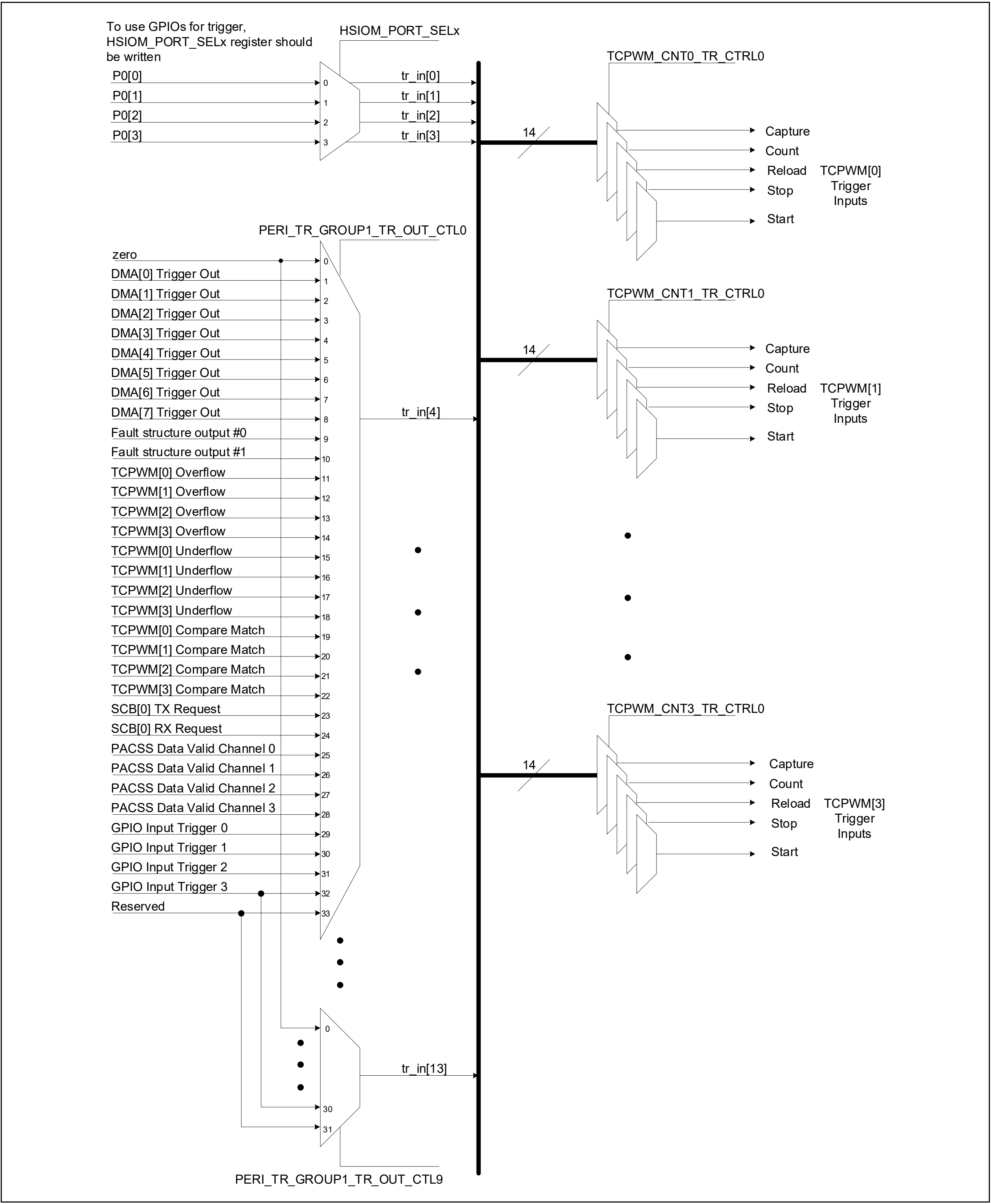


Figure 21-59. Trigger Mux in PSoC™ 4 HV PA

Table 21-49 and **Table 21-50** provide the PSoC™ 4 HV PA TCPWM trigger multiplexers and the multiplexer outputs.

Table 21-49. TCPWM trigger sources

PERI_TR_GROUP1_TR_OUT_CTL x[5:0]	Trigger source
0	Software trigger
1	DMA Channel 0 trigger out
2	DMA Channel 1 trigger out
3	DMA Channel 2 trigger out
4	DMA Channel 3 trigger out
5	DMA Channel 4 trigger out
6	DMA Channel 5 trigger out
7	DMA Channel 6 trigger out
8	DMA Channel 7 trigger out
9	Fault structure output #0
10	Fault structure output #1
11	TCPWM 0 overflow
12	TCPWM 1 overflow
13	TCPWM 2 overflow
14	TCPWM 3 overflow
15	TCPWM 0 underflow
16	TCPWM 1 underflow
17	TCPWM 2 underflow
18	TCPWM 3 underflow
19	TCPWM 0 compare match
20	TCPWM 1 compare match
21	TCPWM 2 compare match
22	TCPWM 3 compare match
23	SCB 0 TX request
24	SCB 0 RX request
25	PACSS data valid channel 0
26	PACSS data valid channel 1
27	PACSS data valid channel 2
28	PACSS data valid channel 3
29	GPIO input trigger 0
30	GPIO input trigger 1
31	GPIO input trigger 2
32	GPIO input trigger 3
33	Reserved

Table 21-50. Trigger Multiplexer Outputs

TCPWM_CNTx_TR_CTRL0	Trigger source
0	Constant ‘0’
1	Constant ‘1’
2	tcpwm.tr_in[0]
3	tcpwm.tr_in[1]
4	tcpwm.tr_in[2]
5	tcpwm.tr_in[3]
6	tcpwm.tr_in[4]
7	tcpwm.tr_in[5]
8	tcpwm.tr_in[6]
9	tcpwm.tr_in[7]
10	tcpwm.tr_in[8]
11	tcpwm.tr_in[9]
12	tcpwm.tr_in[10]
13	tcpwm.tr_in[11]
14	tcpwm.tr_in[12]
15	tcpwm.tr_in[13]

The events derived from these triggers can have different definitions in different modes of the TCPWM block.

- **Reload:** A reload event initializes and starts the counter.
  - In UP counting mode and DOWN counting mode, the count register (TCPWM_CNT_COUNTER) is initialized with ‘0’.
  - In UP/DOWN counting mode, the count register is initialized with ‘1’.
  - In quadrature mode, the reload event acts as a quadrature index event. An index/reload event indicates a completed rotation and can be used to synchronize quadrature decoding.
- **Start:** A start event is used to start counting; it can be used after a stop event or after re-initialization of the counter register to any value by software. Note that the count register is not initialized on this event.
  - In quadrature mode, the start event acts as quadrature phase input phiB, which is explained in detail in **“Quadrature Decoder mode”** on page 259.
- **Count:** A count event causes the counter to increment or decrement, depending on its configuration.
  - In quadrature mode, the count event acts as quadrature phase input phiA.
- **Stop:** A stop event stops the counter from incrementing or decrementing. A start event will start the counting again.
  - In the PWM modes, the stop event acts as a kill event. A kill event disables all the PWM output lines.
- **Capture:** A capture event copies the counter register value to the capture register and capture register value to the buffer capture register. In the PWM modes, the capture event acts as a switch event. It switches the values of the capture/compare and period registers with their buffer counterparts. This feature can be used to modulate the pulse width and frequency.

Notes

- All trigger inputs are synchronized to the high-frequency clock.
- When more than one event occurs in the same counter clock period, one or more events may be missed. This can happen for high-frequency events (frequencies close to the counter frequency) and a timer configuration in which a pre-scaled (divided) counter clock is used.

21.2.4 Output signals

The TCPWM block generates several output signals, as shown in **Figure 21-60**.

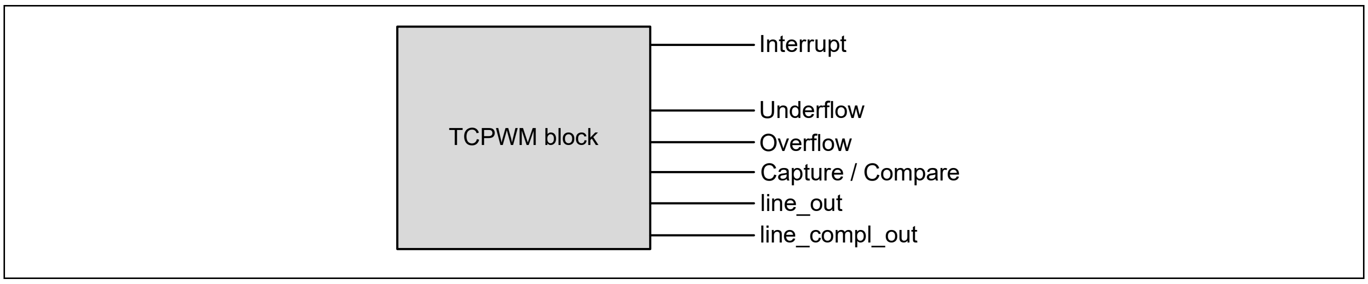


Figure 21-60. TCPWM output signals

21.2.4.1 Signals upon trigger conditions

- Counter generates an internal overflow (OV) condition when counting up and the count register reaches the period value.
- Counter generates an internal underflow (UN) condition when counting down and the count register reaches zero.
- The capture/compare (CC) condition is generated by the TCPWM when the counter is running and one of the following conditions occur:
  - The counter value equals the compare value.
  - A capture event occurs - When a capture event occurs, the TCPWM_CNT_COUNTER register value is copied to the capture register and the capture register value is copied to the buffer capture register.

Note: These signals, when they occur, remain at logic high for two cycles of the high-frequency clock. For reliable operation, the condition that causes this trigger should be less than a quarter of the high-frequency clock. For example, if the high-frequency clock is running at 24 MHz, the condition causing the trigger should occur at a frequency less than 6 MHz.

21.2.4.2 Interrupts

The TCPWM block provides a dedicated interrupt output signal from the counter. An interrupt can be generated for a TC condition or a CC condition. The exact definition of these conditions is mode-specific.

Four registers are used for interrupt handling in this block, as shown in **Table 21-51**.

Table 21-51. Interrupt Register

Interrupt Registers	Bits	Name	Description
TCPWM_CNT_INTR (Interrupt request register)	0	TC	This bit is set to ‘1’, when a terminal count is detected. Write ‘1’ to clear this bit.
	1	CC_MATCH	This bit is set to ‘1’ when the counter value matches capture/compare register value. Write ‘1’ to clear this bit.
TCPWM_CNT_INTR_SET (Interrupt set request register)	0	TC	Write ‘1’ to set the corresponding bit in the interrupt request register. When read, this register reflects the interrupt request register status.
	1	CC_MATCH	Write ‘1’ to set the corresponding bit in the interrupt request register. When read, this register reflects the interrupt request register status.

Table 21-51. Interrupt Register (continued)

Interrupt Registers	Bits	Name	Description
TCPWM_CNT_INTR_MASK (Interrupt mask register)	0	TC	Mask bit for the corresponding TC bit in the interrupt request register.
	1	CC_MATCH	Mask bit for the corresponding CC_MATCH bit in the interrupt request register.
TCPWM_CNT_INTR_MASKED (Interrupt masked request register)	0	TC	Logical AND of the corresponding TC request and mask bits.
	1	CC_MATCH	Logical AND of the corresponding CC_MATCH request and mask bits.

21.2.4.3 Outputs

The TCPWM has two outputs (see **Figure 21-61**), line_out and line_compl_out (complementary of line_out). Note that the OV, UN, and CC conditions can be used to drive line_out and line_compl_out if needed, by configuring the TCPWM_CNT_TR_CTRL2 register (**Table 21-52**).

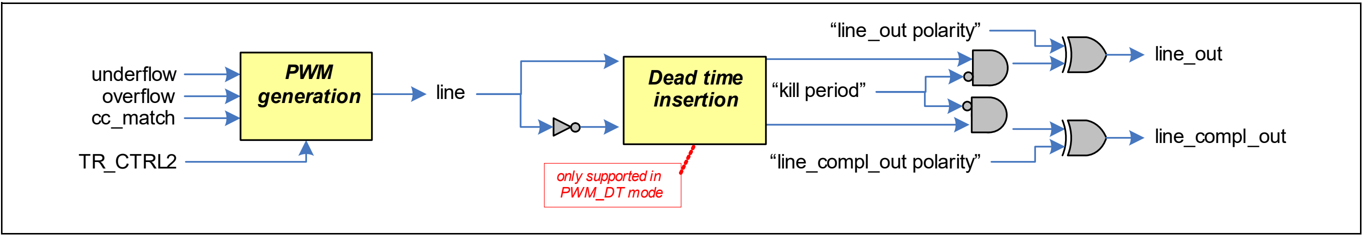


Figure 21-61. Line generation logic

Table 21-52. Configuring output line for OV, UN, and CC Conditions

Field	Bit	Value	Event	Description
CC_MATCH_MODE Default Value = 3	1:0	0	Set line_out to 1	Configures output line on a compare match (CC) event
		1	Clear line_out to 0	
		2	Invert line_out	
		3	No change	
OVERFLOW_MODE Default Value = 3	3:2	0	Set line_out to 1	Configures output line on a overflow (OV) event
		1	Clear line_out to 0	
		2	Invert line_out	
		3	No change	
UNDERFLOW_MODE Default Value = 3	5:4	0	Set line_out to 1	Configures output line on a underflow (UN) event
		1	Clear line_out to 0	
		2	Invert line_out	
		3	No change	

21.2.5 Power modes

The TCPWM block works in Active and Sleep modes. The TCPWM block is powered from V_{CCD}. The configuration registers and other logic are powered in Deep Sleep mode to keep the states of configuration registers. See **Table 21-53** for details.

Table 21-53. Power modes in TCPWM block

Power Mode	Block status
Active	This block is fully operational in this mode with clock running and power switched on.
Sleep	All counter clocks are on, but bus interface cannot be accessed.
Deep Sleep	In this mode, the power to this block is still on but no bus clock is provided; hence, the logic is not functional. All the configuration registers will keep their state.

21.3 Operation modes

The counter block can function in six operational modes, as shown in **Table 21-54**. The MODE [26:24] field of the counter control register (TCPWM_CNTx_CTRL) configures the counter in the specific operational mode.

Table 21-54. Operational mode configuration

Mode	MODE Field [26:24]	Description
Timer	000	Implements a timer or counter. The counter increments or decrements by ‘1’ at every counter clock cycle in which a count event is detected.
Capture	010	Implements a timer or counter with capture input. The counter increments or decrements by ‘1’ at every counter clock cycle in which a count event is detected. When a capture event occurs, the counter value copies into the capture register.
Quadrature Decoder	011	Implements a quadrature decoder, where the counter is decremented or incremented, based on two phase inputs according to the selected (X1, X2 or X4) encoding scheme.
PWM	100	Implements edge/center-aligned PWMs with an 8-bit clock prescaler and buffered compare/period registers.
PWM-DT	101	Implements edge/center-aligned PWMs with configurable 8-bit dead time (on both outputs) and buffered compare/period registers.
PWM-PR	110	Implements a pseudo-random PWM using a 16-bit linear feedback shift register (LFSR).

The counter can be configured to count up, down, and up/down by setting the UP_DOWN_MODE[17:16] field in the TCPWM_CNT_CTRL register, as shown in **Table 21-55**.

Table 21-55. Counting mode configuration

Counting Modes	UP_DOWN_ MODE[17:16]	Description
UP Counting Mode	00	Increments the counter until the period value is reached. A Terminal Count (TC) condition is generated when the counter reaches the period value.
DOWN Counting Mode	01	Decrements the counter from the period value until 0 is reached. A TC condition is generated when the counter reaches ‘0’.
UP/DOWN Counting Mode 0	10	Increments the counter until the period value is reached, and then decrements the counter until ‘0’ is reached. A TC condition is generated only when ‘0’ is reached.
UP/DOWN Counting Mode 1	11	Similar to up/down counting mode 0 but a TC condition is generated when the counter reaches ‘0’ and when the counter value reaches the period value.

21.3.1 Timer mode

The timer mode is commonly used to measure the time of occurrence of an event or to measure the time difference between two events.

21.3.1.1 Block diagram

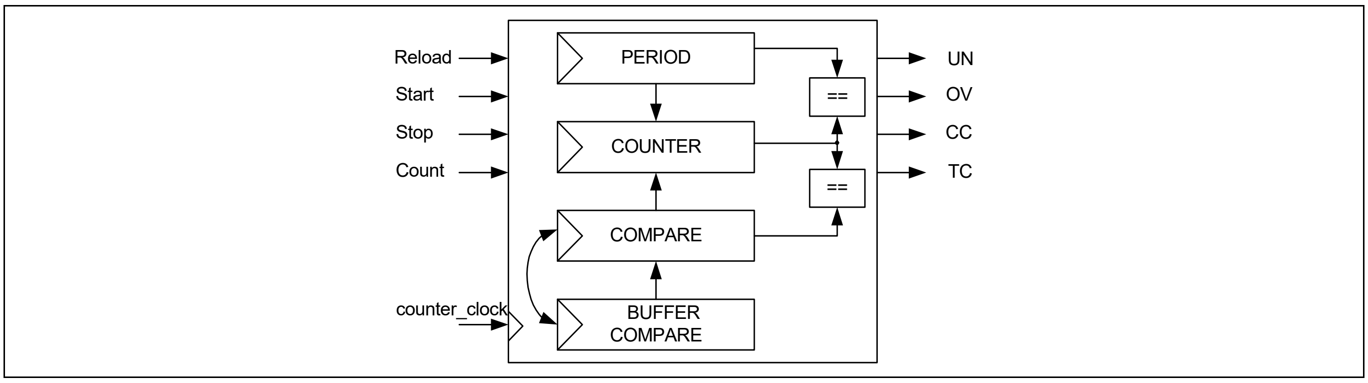


Figure 21-62. Timer mode block diagram

21.3.1.2 How it works

The timer can be configured to count in up, down, and up/down counting modes. It can also be configured to run in either continuous mode or one-shot mode. The following explains the working of the timer:

- The timer is an up, down, and up/down counter.
  - The current count value is stored in the count register (TCPWM_CNTx_COUNTER).
    Note: It is not recommended to write values to this register while the counter is running.
  - The period value for the timer is stored in the period register.
- The counter is re-initialized in different counting modes as follows:
  - In the up counting mode, after the count reaches the period value, the count register is automatically reloaded with 0.
  - In the down counting mode, after the count register reaches zero, the count register is reloaded with the value in the period register.
  - In the up/down counting modes, the count register value is not updated upon reaching the terminal values. Instead the direction of counting changes when the count value reaches 0 or the period value.
- The CC condition is generated when the count register value equals the compare register value. Upon this condition, the compare register and buffer compare register switch their values if enabled by the AUTO_RELOAD_CC bit-field of the counter control (TCPWM_CNT_CTRL) register. This condition can be used to generate an interrupt request.

Figure 21-63 shows the timer operational mode of the counter in four different counting modes. The period register contains the maximum counter value.

- In the up counting mode, a period value of A results in A + 1 counter cycles (0 to A).
- In the down counting mode, a period value of A results in A + 1 counter cycles (A to 0).
- In the two up/down counting modes (0 and 1), a period value of A results in 2 × A counter cycles (0 to A and back to 0).

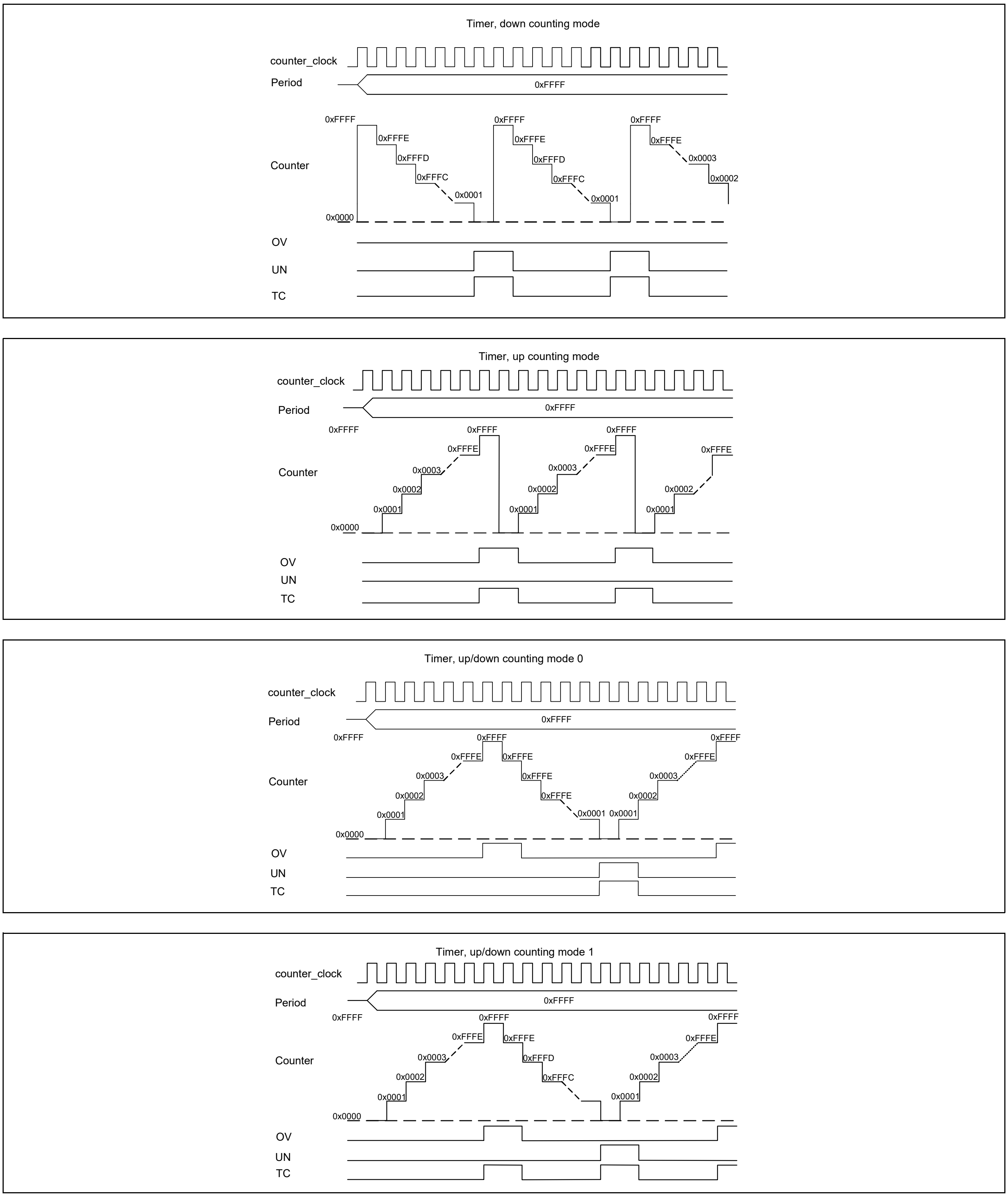


Figure 21-63. Timing diagram for timer in multiple counting modes

Note: The OV and UN signals remain at logic high for two cycles of the high-frequency clock, as explained in **"Signals upon trigger conditions"** on page 250. The figures in this chapter assume that high-frequency clock and counter clock are the same.

21.3.1.3 Configuring counter for Timer mode

The steps to configure the counter for Timer mode of operation and the affected register bits are as follows.

1. Disable the counter by writing ‘0’ to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Timer mode by writing ‘000’ to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNT_PERIOD register.
4. Set the 16-bit compare value in the TCPWM_CNT_CC register and the buffer compare value in the TCPWM_CNT_CC_BUFF register.
5. Set AUTO_RELOAD_CC field of the TCPWM_CNT_CTRL register, if required to switch values at every CC condition.
6. Set clock prescaling by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in **Table 21-48**.
7. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNT_CTRL register, as shown in **Table 21-55**.
8. The timer can be configured to run either in continuous mode or one-shot mode by writing 0 or 1, respectively to the ONE_SHOT[18] field of TCPWM_CNT_CTRL.
9. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Stop, Capture, and Count).
10. Set the TCPWM_CNT_TR_CTRL1 register to select the edge of the trigger that causes the event (Reload, Start, Stop, Capture, and Count).
11. If required, set the interrupt upon TC or CC condition, as shown in **“Interrupts”** on page 250.
12. Enable the counter by writing ‘1’ to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if the hardware start signal is not enabled.

21.3.2 Capture mode

In the capture mode, the counter value can be captured at any time either through a firmware write to command register (TCPWM_CMD) or a capture trigger input. This mode is used for period and pulse width measurement.

21.3.2.1 Block diagram

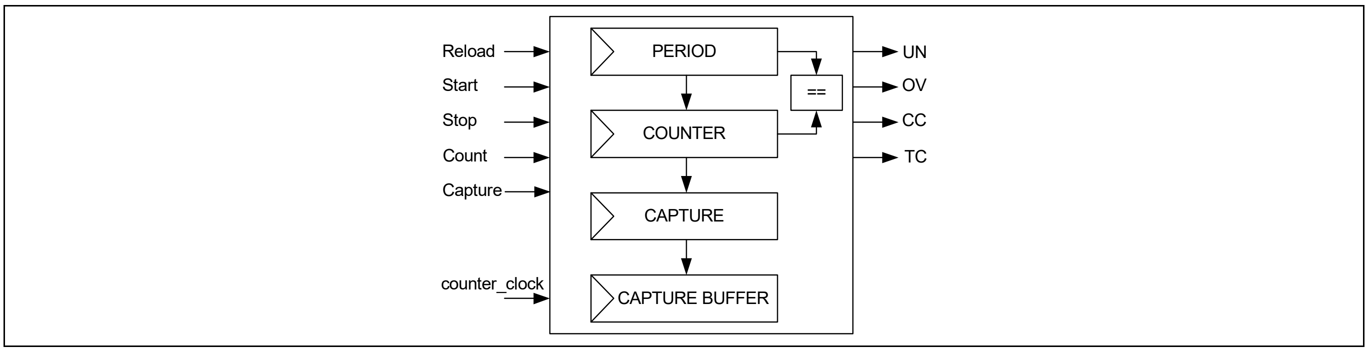


Figure 21-64. Capture mode block diagram

21.3.2.2 How it works

The counter can be set to count in up, down, and up/down counting modes by configuring the UP_DOWN_MODE[17:16] bit-field of the counter control register (TCPWM_CNT_CTRL).

Operation in capture mode occurs as follows:

- During a capture event, generated either by hardware or software, the current count register value is copied to the capture register (TCPWM_CNT_CC) and the capture register value is copied to the buffer capture register (TCPWM_CNT_CC_BUFF).
- A pulse on the CC output signal is generated when the counter value is copied to the capture register. This condition can also be used to generate an interrupt request.

Figure 21-65 illustrates the capture behavior in the up counting mode.

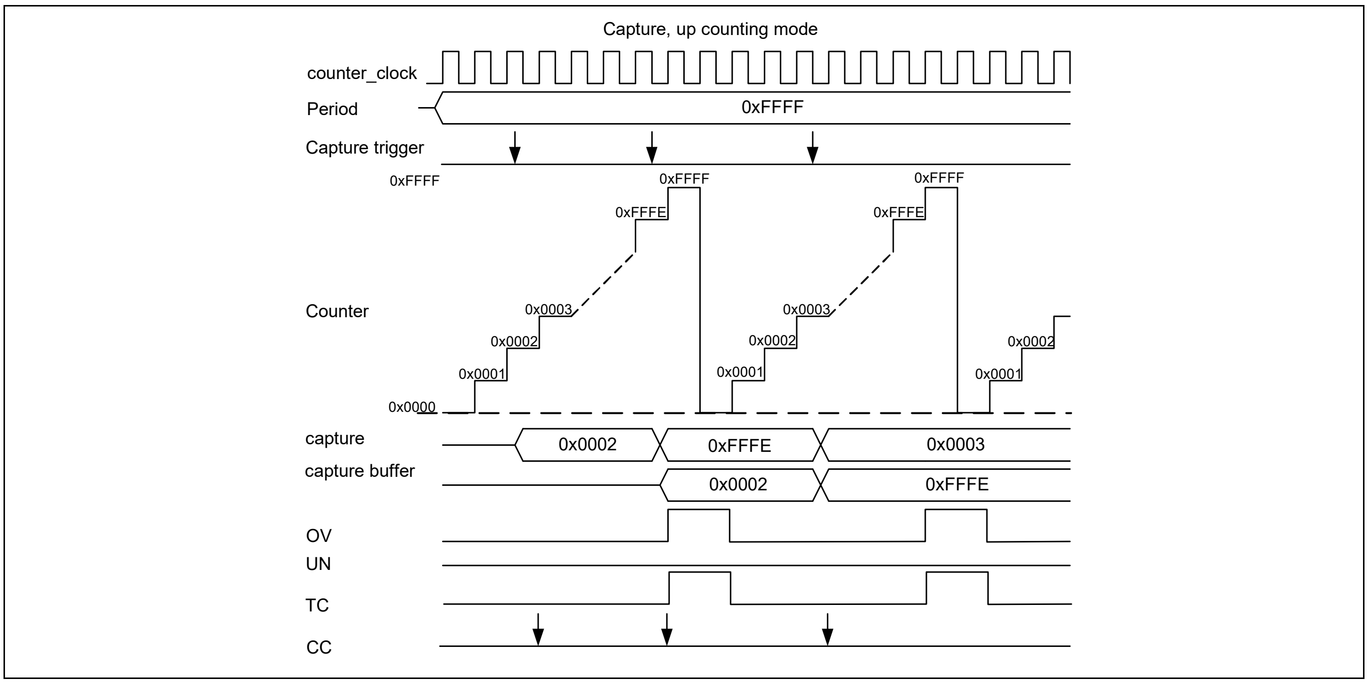


Figure 21-65. Timing diagram of counter in Capture Mode, Up Counting Mode

In **Figure 21-65**, observe that:

- The period register contains the maximum count value.
- Internal overflow (OV) and TC conditions are generated when the counter reaches the period value.
- A capture event is only possible at the edges or through software. Use trigger control register 1 to configure the edge detection.
- Multiple capture events in a single clock cycle are handled as:
  - Even number of capture events - no event is observed
  - Odd number of capture events - single event is observed

This happens when the capture signal frequency is greater than the counter_clock frequency.

21.3.2.3 Configuring counter for Capture mode

The steps to configure the counter for Capture mode operation and the affected register bits are as follows.

1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Capture mode by writing '010' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNT_PERIOD register.
4. Set clock prescaling by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in **Table 21-48**.
5. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNT_CTRL register, as shown in **Table 21-55**.
6. Counter can be configured to run either in continuous mode or one-shot mode by writing 0 or 1, respectively to the ONE_SHOT[18] field of the TCPWM_CNT_CTRL register.
7. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Stop, Capture, and Count).
8. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Stop, Capture, and Count).
9. If required, set the interrupt upon TC or CC condition, as shown in **"Interrupts"** on page 250.
10. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if the hardware start signal is not enabled.

21.3.3 Quadrature Decoder mode

Quadrature decoders are used to determine speed and position of a rotary device (such as servo motors, volume control wheels, and PC mice). The quadrature encoder signals are used as phiA and phiB inputs to the decoder.

21.3.3.1 Block diagram

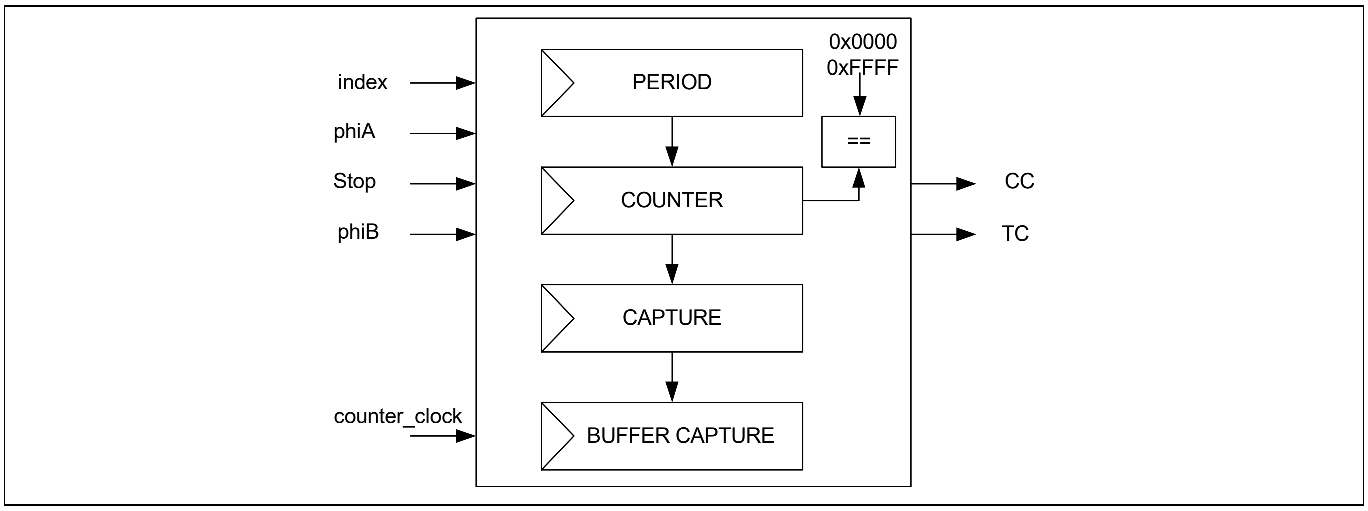


Figure 21-66. Quadrature Mode block diagram

21.3.3.2 How it works

Quadrature decoding only runs on counter_clock. It can operate in three sub-modes: X1, X2, and X4 modes. These encoding modes can be controlled by the QUADRATURE_MODE[21:20] field of the counter control register (TCPWM_CNT_CTRL). This mode uses double buffered capture registers.

The Quadrature mode operation occurs as follows:

- Quadrature phases phiA and phiB: Counting direction is determined by the phase relationship between phiA and phiB. These phases are connected to the count and the start trigger inputs, respectively as hardware input to the decoder.
- Quadrature index signal: This is connected to the reload signal as a hardware input. This event generates a TC condition, as shown in **Figure 21-67**.
- On TC, the counter is set to 0x0000 (in the up counting mode) or to the period value (in the down counting mode).
- **Note:** The down counting mode is recommended to be used with a period value of 0x8000 (the mid-point value).
- A pulse on CC output signal is generated when the count register value reaches 0x0000 or 0xFFFF. On a CC condition, the count register is set to the period value (0x8000 in this case).
- On TC or CC condition:
  - Count register value is copied to the capture register.
  - Capture register value is copied to the buffer capture register.
  - This condition can be used to generate an interrupt request.
- The value in the capture register can be used to determine which condition caused the event and whether:
  - A counter underflow occurred (value 0)
  - A counter overflow occurred (value 0xFFFF)
  - An index/TC event occurred (value is not equal to either 0 or 0xFFFF)
- The DOWN bit field of counter status (TCPWM_CNTx_STATUS) register can be read to determine the current counting direction. Value ‘0’ indicates a previous increment operation and value ‘1’ indicates previous decrement operation. **Figure 21-67** illustrates quadrature behavior in the X1 encoding mode.
  - A positive edge on phiA increments the counter when phiB is ‘0’ and decrements the counter when phiB is ‘1’.
  - The count register is initialized with the period value on an index/reload event.
  - Terminal count is generated when the counter is initialized by index event. This event can be used to generate an interrupt.
  - When the count register reaches 0xFFFF (the maximum count register value), the count register value is copied to the capture register and the count register is initialized with period value (0x8000 in this case).

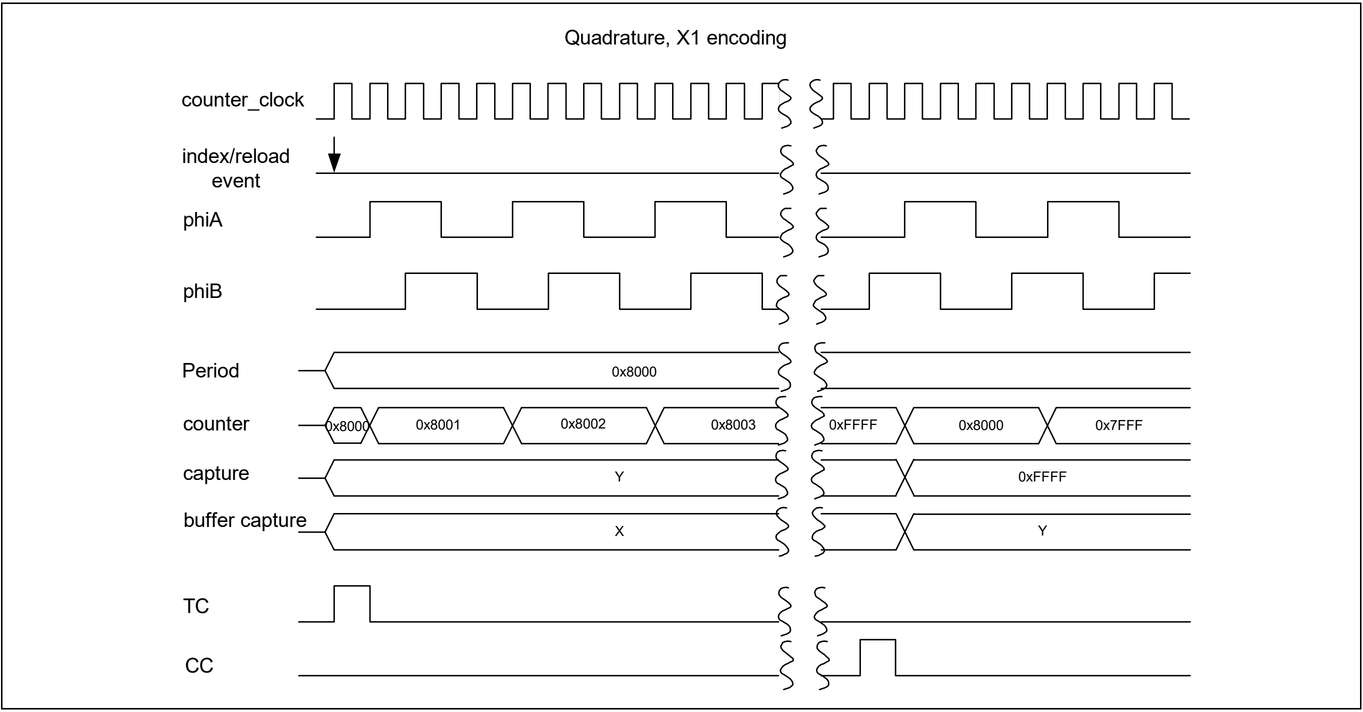


Figure 21-67. Timing diagram for Quadrature Mode, X1 Encoding

The quadrature phases are detected on the counter_clock. Within a single counter_clock period, the phases should not change value more than once. The X2 and X4 quadrature encoding modes count twice and four times as fast as the X1 encoding mode.

Figure 21-68 illustrates the quadrature mode behavior in the X2 and X4 encoding modes.

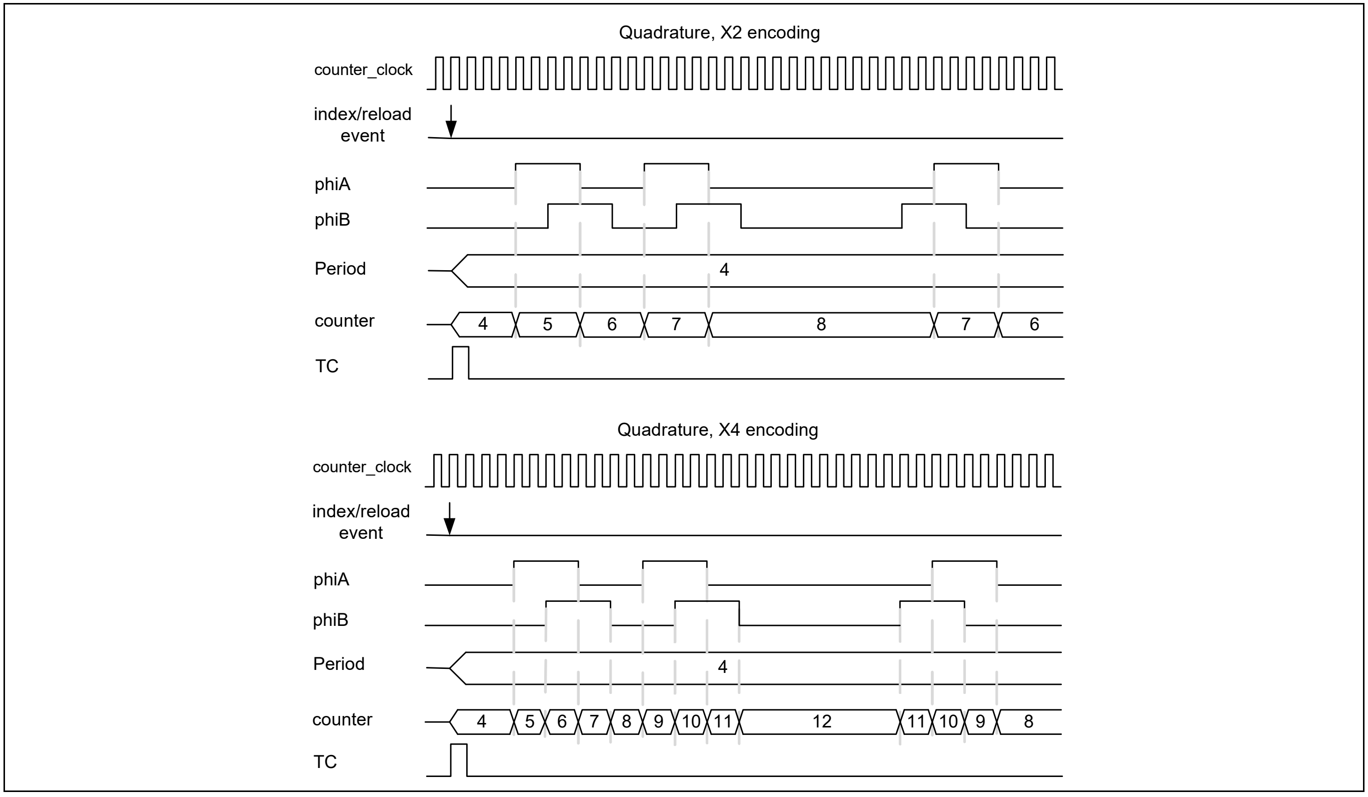


Figure 21-68. Timing diagram for Quadrature Mode, X2 and X4 Encoding

21.3.3.3 Configuring counter for Quadrature mode

The steps to configure the counter for quadrature mode of operation and the affected register bits are as follows.

1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Quadrature mode by writing '011' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNT_PERIOD register.
4. Set the required encoding mode by writing to the QUADRATURE_MODE[21:20] field of the TCPWM_CNT_CTRL register.
5. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Index and Stop).
6. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Index and Stop).
7. If required, set the interrupt upon TC or CC condition, as shown in **"Interrupts"** on page 250.
8. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register.

21.3.4 Pulse Width Modulation mode

The PWM mode is also called the Digital Comparator mode. The comparison output is a PWM signal whose period depends on the period register value and duty cycle depends on the compare and period register values.

PWM period = (period value/counter clock frequency) in left- and right-aligned modes

PWM period = (2 × (period value/counter clock frequency)) in center-aligned mode

Duty cycle = (compare value/period value) in left- and right-aligned modes

Duty cycle = ((period value-compare value)/period value) in center-aligned mode

21.3.4.1 Block diagram

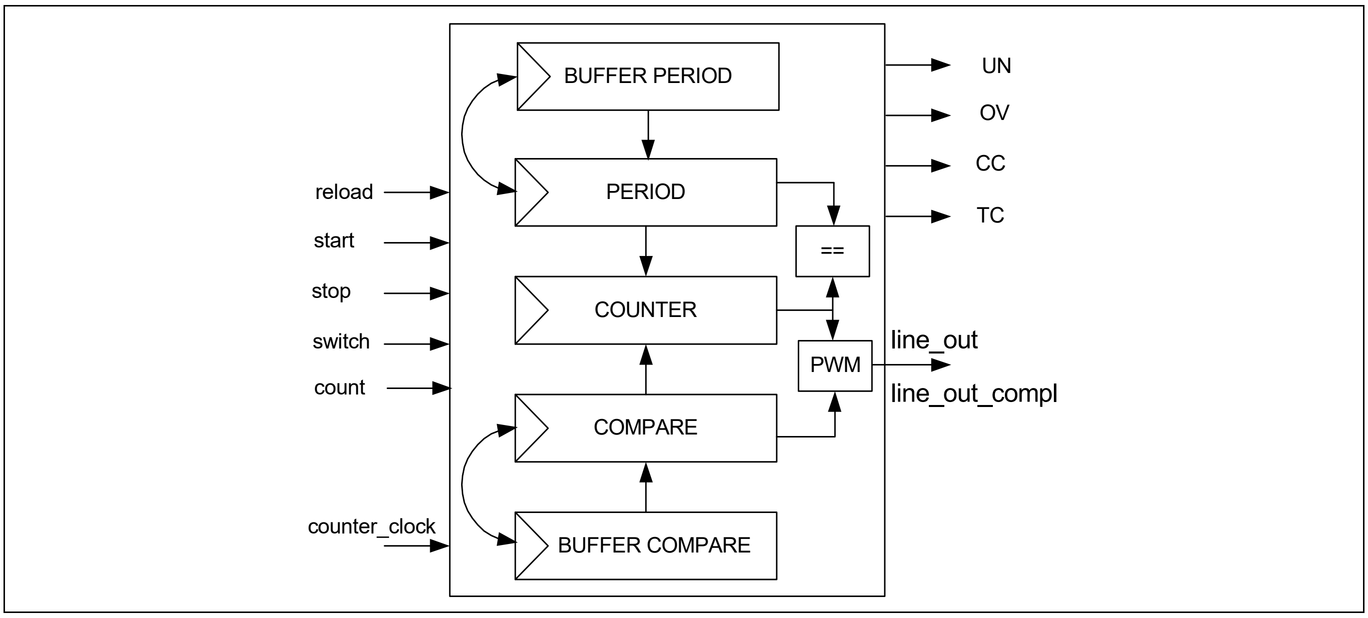


Figure 21-69. PWM Mode block diagram

21.3.4.2 How it works

The PWM mode can output left, right, center, or asymmetrically aligned PWM signals. The desired output alignment is achieved by using the counter's up, down, and up/down counting modes selected using UP_DOWN_MODE [17:16] bits in the TCPWM_CNT_CTRL register, as shown in **Table 21-55**.

This CC signal along with OV and UN signals control the PWM output line. The signals can toggle the output line or set it to a logic '0' or '1' by configuring the TCPWM_CNT_TR_CTRL2 register. By configuring how the signals impact the output line, the desired PWM output alignment can be obtained.

The recommended way to modify the duty cycle is:

- The buffer period register and buffer compare register are updated with new values.
- On TC, the period and compare registers are automatically updated with the buffer period and buffer compare registers when there is an active switch event. The AUTO_RELOAD_CC and AUTO_RELOAD_PERIOD fields of the counter control register are set to '1'. When a switch event is detected, it is remembered until the next TC event. Pass through signal (selected during event detection setting) cannot trigger a switch event.
- Updates to the buffer period register and buffer compare register should be completed before the next TC with an active switch event; otherwise, switching does not reflect the register update, as shown in **Figure 21-71**.

In the center-aligned mode, the output line is set to '0' at Terminal Count and toggled at the CC condition

At the reload event, the count register is initialized and starts counting in the appropriate mode. At every count, the count register value is compared with compare register value to generate the CC signal on match.

Figure 21-70 illustrates center-aligned PWM with buffered period and compare registers (up/down counting mode 0).

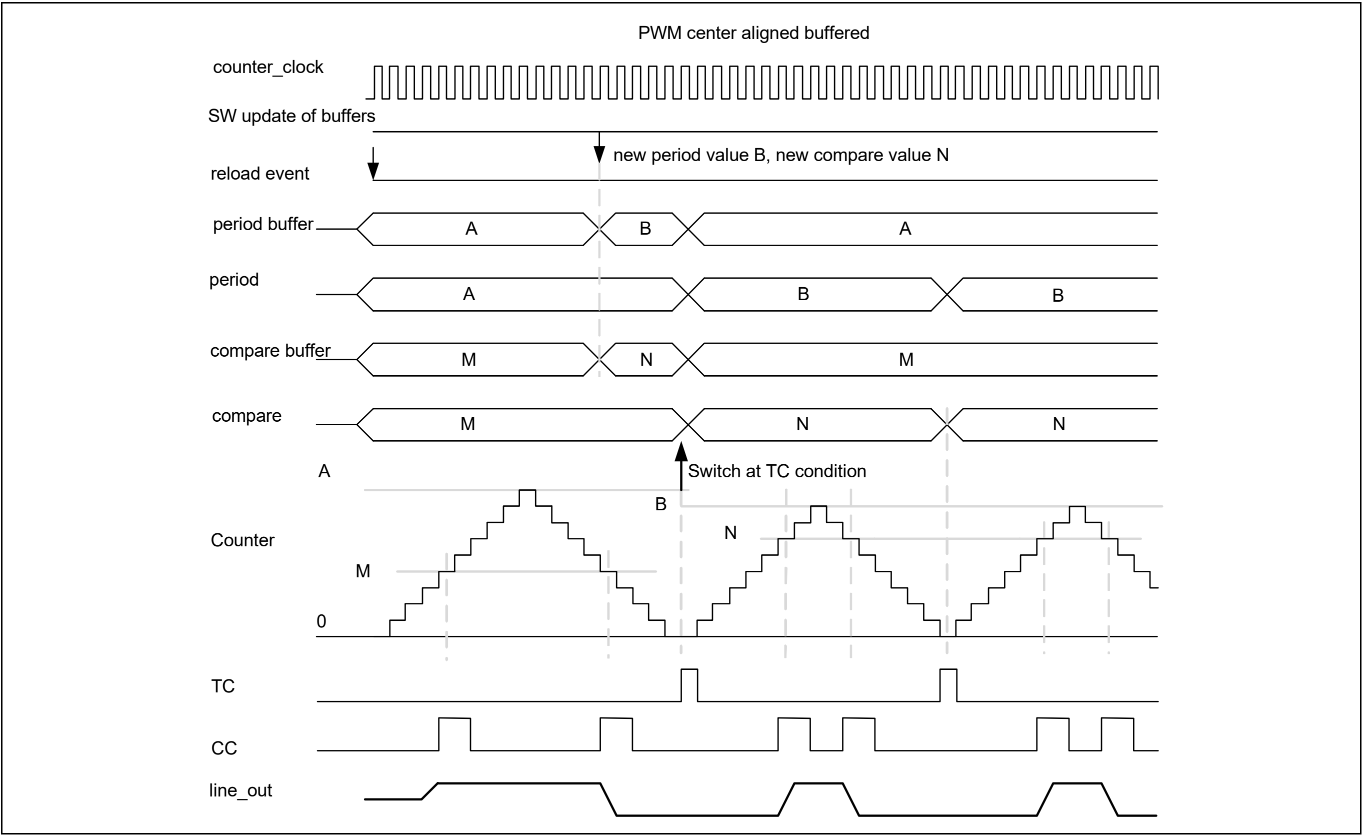


Figure 21-70. Timing diagram for center aligned PWM

Figure 21-70 illustrates center-aligned PWM with software generated switch events:

- Software generates a switch event only after both the period buffer and compare buffer registers are updated.
- Because the updates of the second PWM pulse come late (after the terminal count), the first PWM pulse is repeated.
- Note that the switch event is automatically cleared by hardware at TC after the event takes effect.

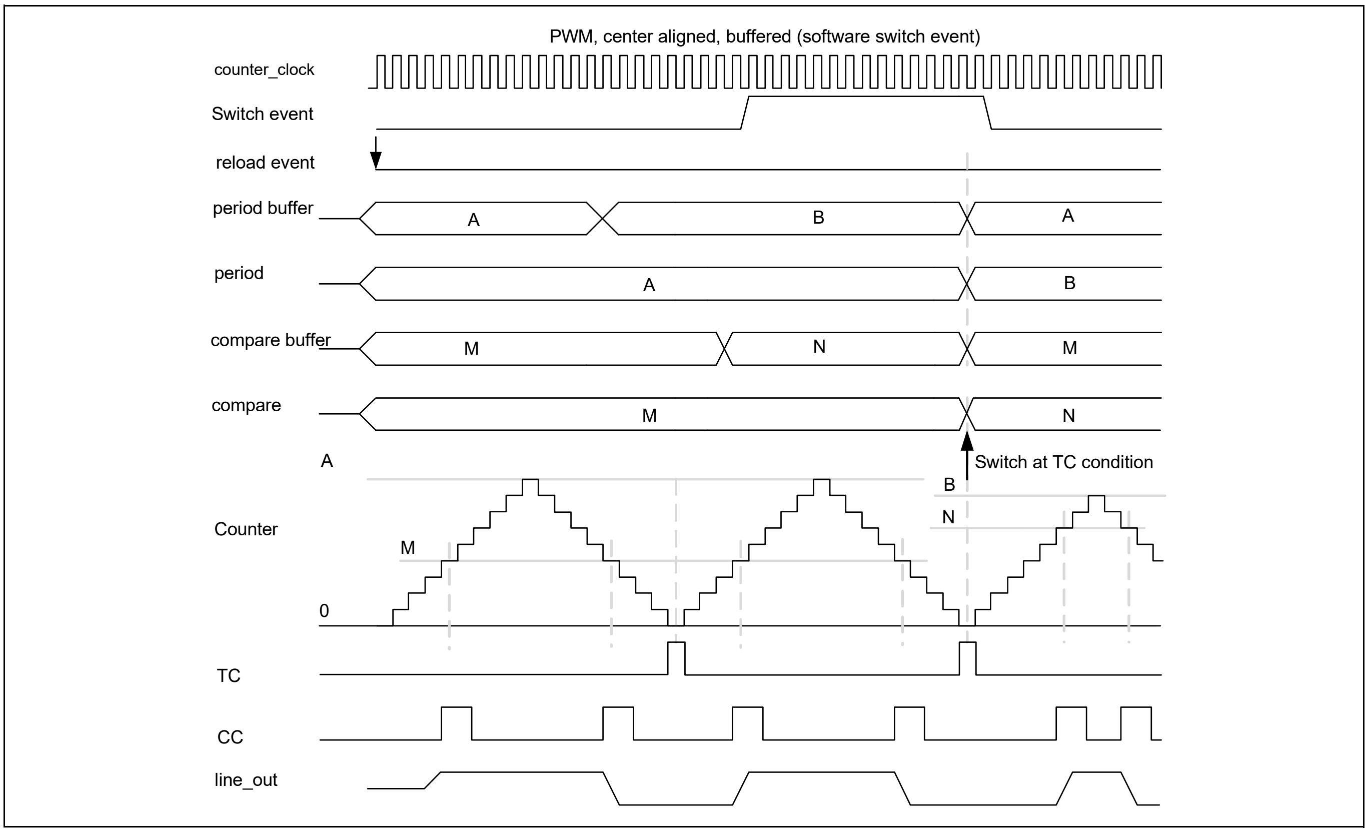


Figure 21-71. Timing diagram for center aligned PWM (software switch event)

21.3.4.3 Other configurations

- For asymmetric PWM, the up/down counting mode 1 should be used. This causes a TC when the counter reaches either '0' or the period value. To create an asymmetric PWM, the compare register is changed at every TC (when the counter reaches either '0' or the period value), whereas the period register is only changed at every other TC (only when the counter reaches '0').
- For left-aligned PWM, use the up counting mode; configure the OV condition to set output line to '1' and CC condition to reset the output line to '0'. See **Table 21-52**.
- For right-aligned PWM, use the down counting mode; configure UN condition to reset output line to '0' and CC condition to set the output line to '1'. See **Table 21-52**.

21.3.4.4 Kill feature

The kill feature gives the ability to disable both output lines immediately. This event can be programmed to stop the counter by modifying the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the counter control register, as shown in **Table 21-56**.

Table 21-56. Field setting for stop on kill feature

PWM_STOP_ON_KILL Field	Comments
0	The kill trigger temporarily blocks the PWM output line but the counter is still running.
1	The kill trigger temporarily blocks the PWM output line and the counter is also stopped.

A kill event can be programmed to be asynchronous or synchronous, as shown in **Table 21-57**.

Table 21-57. Field Setting for Synchronous/Asynchronous Kill

PWM_SYNC_KILL Field	Comments
0	An asynchronous kill event lasts as long as it is present. This event requires pass through mode.
1	A synchronous kill event disables the output lines until the next TC event. This event requires rising edge mode.

In the synchronous kill, PWM cannot be started before the next TC. To restart the PWM immediately after kill input is removed, kill event should be asynchronous (see **Table 21-57**). The generated stop event disables both output lines. In this case, the reload event can use the same trigger input signal but should be used in falling edge detection mode.

21.3.4.5 Configuring Counter for PWM Mode

The steps to configure the counter for the PWM mode of operation and the affected register bits are as follows.

1. Disable the counter by writing ‘0’ to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select PWM mode by writing ‘100’ to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set clock prescaling by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in **Table 21-48**.
4. Set the required 16-bit period in the TCPWM_CNT_PERIOD register and the buffer period value in the TCPWM_CNT_PERIOD_BUFF register to switch values, if required.
5. Set the 16-bit compare value in the TCPWM_CNT_CC register and buffer compare value in the TCPWM_CNT_CC_BUFF register to switch values, if required.
6. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNT_CTRL register to configure left-aligned, right-aligned, or center-aligned PWM, as shown in **Table 21-55**.
7. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNT_CTRL register as required.
8. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, Switch, and Count).
9. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, Switch, and Count).
10. line_out and line_out_compl can be controlled by the TCPWM_CNT_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
11. If required, set the interrupt upon TC or CC condition, as shown in **“Interrupts”** on page 250.
12. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if the hardware start signal is not enabled.

21.3.5 Pulse width modulation with Dead Time mode

Dead time is used to delay the transitions of both ‘line_out’ and ‘line_out_compl’ signals. It separates the transition edges of these two signals by a specified time interval. Two complementary output lines ‘dt_line’ and ‘dt_line_compl’ are derived from these two lines. During the dead band period, both compare output and complement compare output are at logic ‘0’ for a fixed period. The dead band feature allows the generation of two non-overlapping PWM pulses. A maximum dead time of 255 clocks can be generated using this feature.

21.3.5.1 Block diagram

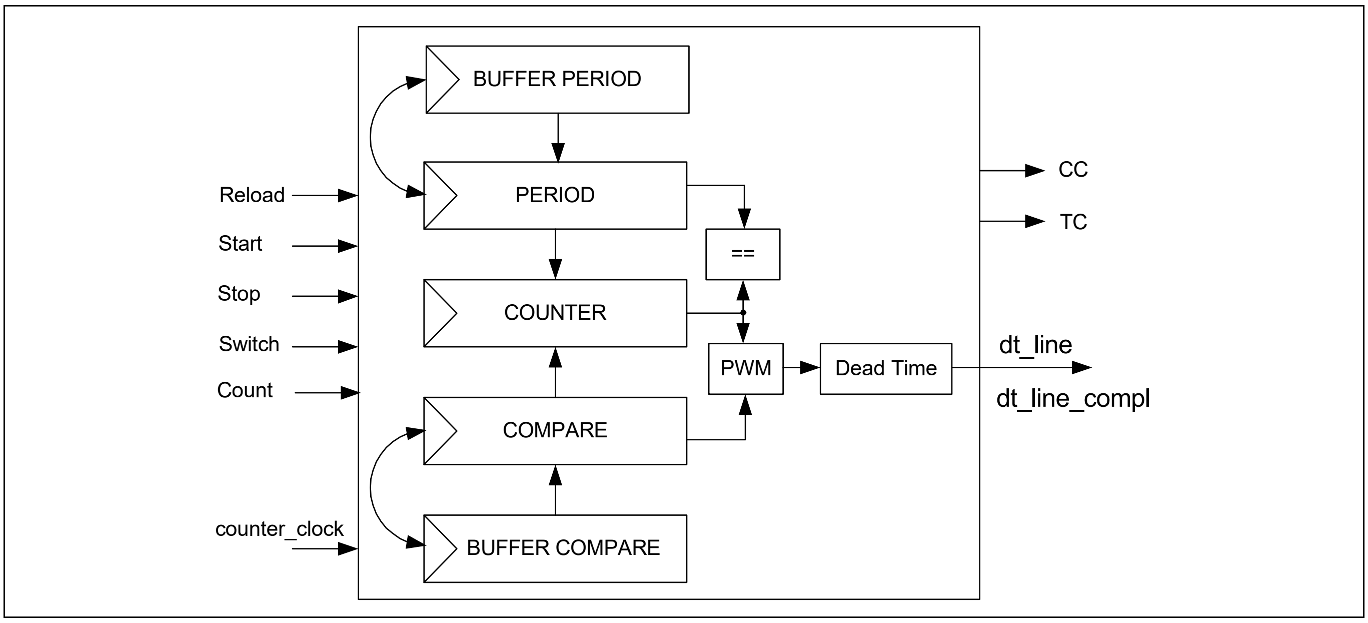


Figure 21-72. PWM-DT mode block diagram

21.3.5.2 How it works

The PWM operation with Dead Time mode occurs as follows:

- On the rising edge of the PWM line_out, depending upon UN, OV, and CC conditions, the dead time block sets the dt_line and dt_line_compl to ‘0’.
- The dead band period is loaded and counted for the period configured in the register.
- When the dead band period is complete, dt_line is set to ‘1’.
- On the falling edge of the PWM line_out depending upon UN, OV, and CC conditions, the dead time block sets the dt_line and dt_line_compl to ‘0’.
- The dead band period is loaded and counted for the period configured in the register.
- When the dead band period has completed, dt_line_compl is set to ‘1’.
- A dead band period of zero has no effect on the dt_line and is the same as line_out.
- When the duration of the dead time equals or exceeds the width of a pulse, the pulse is removed.

This mode follows PWM mode and supports the following features available with that mode:

- Various output alignment modes
- Two complementary output lines, dt_line and dt_line_compl, derived from PWM line_out and line _out_compl, respectively
  - Stop/kill event with synchronous and asynchronous modes
  - Conditional switch event for compare and buffer compare registers and period and buffer period registers

This mode does not support clock prescaling.

Figure 21-73 illustrates how the complementary output lines dt_line and dt_line_compl are generated from the PWM output line, line_out.

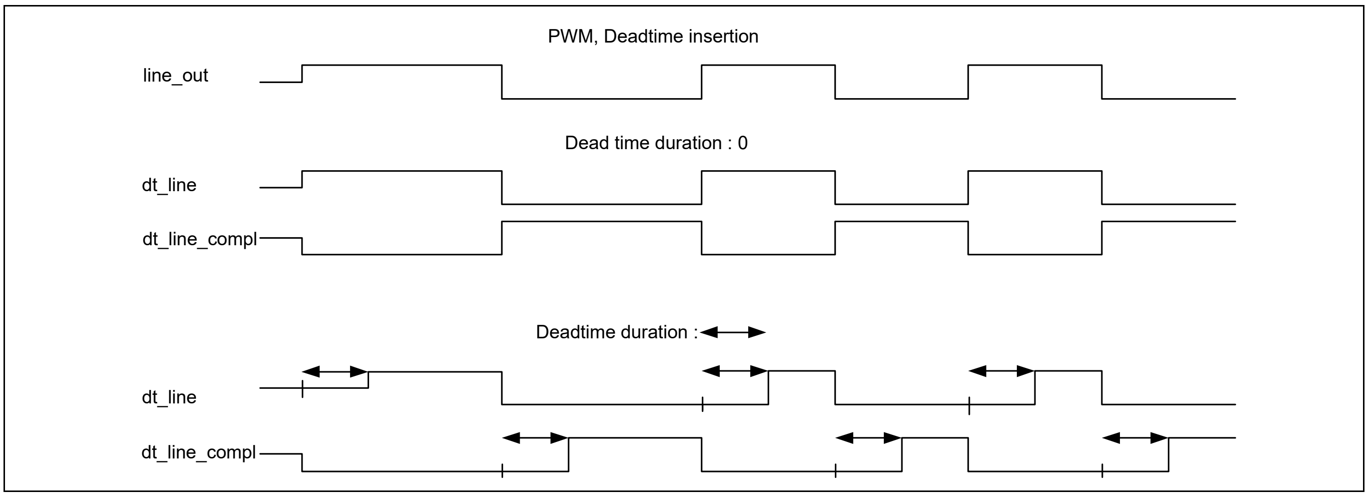


Figure 21-73. Timing diagram for PWM, with and without Dead Time

21.3.5.3 Configuring counter for PWM with Dead Time mode

The steps to configure the counter for PWM with Dead Time mode of operation and the affected register bits are as follows:

1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select PWM with Dead Time mode by writing '101' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required dead time by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in **Table 21-48**.
4. Set the required 16-bit period in the TCPWM_CNT_PERIOD register and the buffer period value in the TCPWM_CNT_PERIOD_BUFF register to switch values, if required.
5. Set the 16-bit compare value in the TCPWM_CNT_CC register and the buffer compare value in the TCPWM_CNT_CC_BUFF register to switch values, if required.
6. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNT_CTRL register to configure left-aligned, right-aligned, or center-aligned PWM, as shown in **Table 21-55**.
7. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNT_CTRL register as required, as shown in the **"Pulse Width Modulation mode"** on page 263.
8. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, Switch, and Count).
9. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, Switch, and Count).
10. dt_line and dt_line_compl can be controlled by the TCPWM_CNT_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
11. If required, set the interrupt upon TC or CC condition, as shown in **"Interrupts"** on page 250.
12. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if hardware start signal is not enabled.

21.3.6 Pulse width modulation Pseudo-Random mode

This mode uses the linear feedback shift register (LFSR). LFSR is a shift register whose input bit is a linear function of its previous state.

21.3.6.1 Block diagram

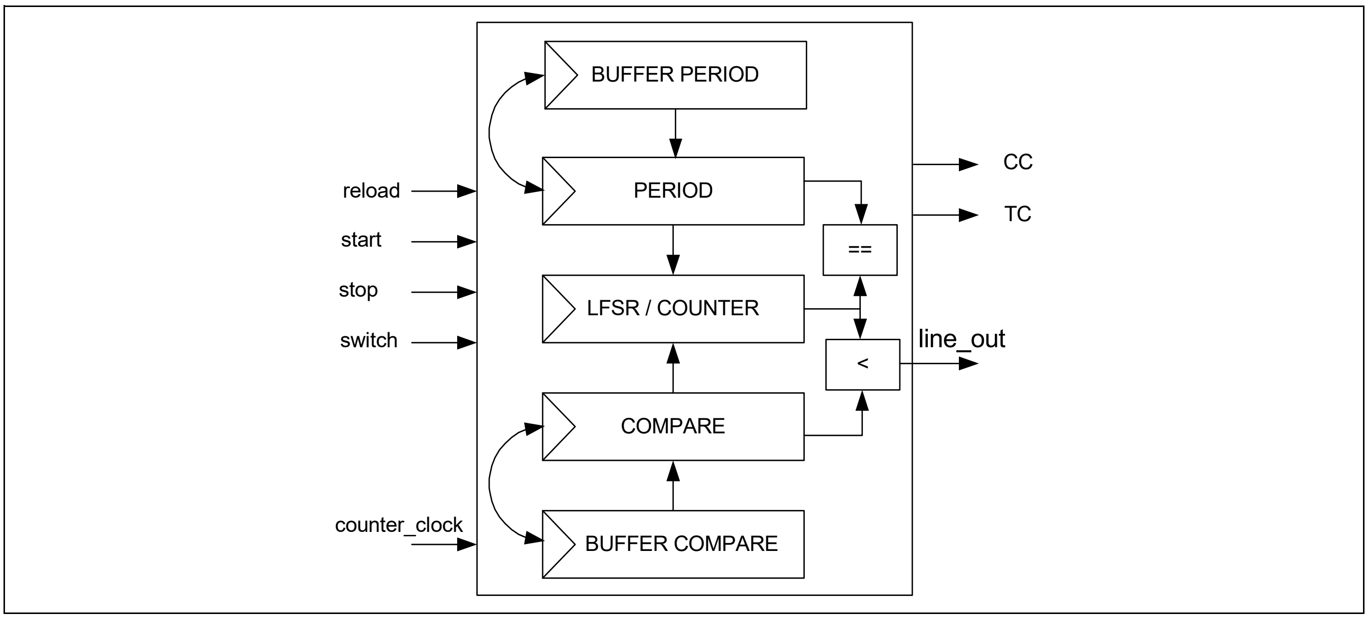


Figure 21-74. PWM-PR mode block diagram

21.3.6.2 How it works

The counter register is used to implement LFSR with the polynomial: $x^{16} + x^{14} + x^{13} + x^{11} + 1$, as shown in **Figure 21-75**. It generates all the numbers in the range [1, 0xFFFF] in a pseudo-random sequence. Note that the counter register should be initialized with a non-zero value.

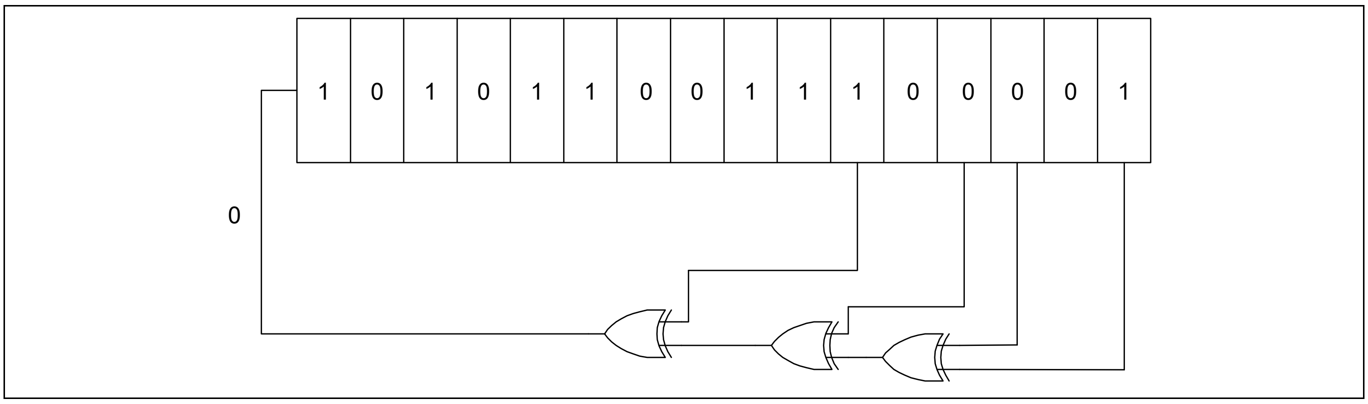


Figure 21-75. Pseudo-random sequence generation using counter register

The following steps describe the process:

- The PWM output line, 'line_out', is driven with '1' when the lower 15-bit value of the counter register is smaller than the value in the compare register (when counter[14:0] < compare[15:0]). A compare value of '0x8000' or higher always results in a '1' on the PWM output line. A compare value of '0' always results in a '0' on the PWM output line.
- A reload event behaves similar to a start event; however, it does not initialize the counter.
- Terminal count is generated when the counter value equals the period value. LFSR generates a predictable pattern of counter values for a certain initial value. This predictability can be used to calculate the counter value after a certain amount of LFSR iterations 'n'. This calculated counter value can be used as a period value and the TC is generated after 'n' iterations.
- At TC, a switch/capture event conditionally switches the compare and period register pairs (based on the AUTO_RELOAD_CC and AUTO_RELOAD_PERIOD fields of the counter control register).
- A kill event can be programmed to stop the counter as described in previous sections.
- One shot mode can be configured by setting the ONE_SHOT field of the counter control register. At terminal count, the counter is stopped by hardware.
- In this mode, underflow, overflow, and trigger condition events do not occur.
- CC condition occurs when the counter is running and its value equals compare value. **Figure 21-76** illustrates pseudo-random noise behavior.
- A compare value of 0x4000 results in 50 percent duty cycle (only the lower 15 bits of the 16-bit counter are used to compare with the compare register value).

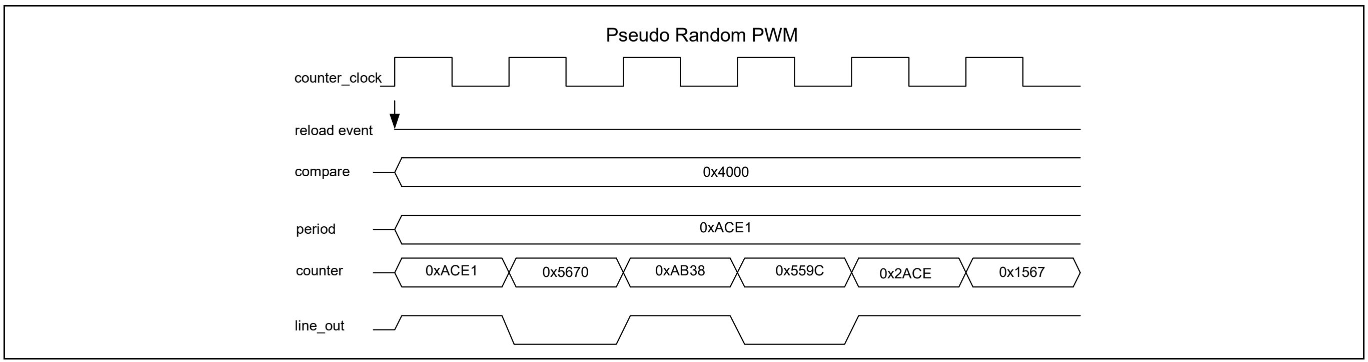


Figure 21-76. Timing diagram for pseudo-random PWM

A capture/switch input signal may switch the values between the compare and compare buffer registers and the period and period buffer registers. This functionality can be used to modulate between two different compare values using a trigger input signal to control the modulation.

Note: Capture/switch input signal can only be triggered by an edge (rising, falling, or both). This input signal is remembered until the next terminal count.

21.3.6.3 Configuring counter for pseudo-random PWM mode

The steps to configure the counter for pseudo-random PWM mode of operation and the affected register bits are as follows.

1. Disable the counter by writing ‘0’ to COUNTER_ENABLED of the TCPWM_CTRL register.
2. Select pseudo-random PWM mode by writing ‘110’ to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required period (16 bit) in the TCPWM_CNT_PERIOD register and buffer period value in the TCPWM_CNT_PERIOD_BUFF register to switch values, if required.
4. Set the 16-bit compare value in the TCPWM_CNT_CC register and the buffer compare value in the TCPWM_CNT_CC_BUFF register to switch values.
5. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNT_CTRL register as required.
6. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, and Switch).
7. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, and Switch).
8. line_out and line_out_compl can be controlled by the TCPWM_CNT_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
9. If required, set the interrupt upon TC or CC condition, as shown in **“Interrupts”** on page 250.
10. Enable the counter by writing ‘1’ to the COUNTER_ENABLED field of the TCPWM_CTRL register.

21.4 TCPWM registers

Table 21-58. List of TCPWM registers

Register	Comment	Features
TCPWM_CTRL	TCPWM Control Register	Enables the counter block
TCPWM_CMD	TCPWM Command Register	Generates software events
TCPWM_INTR_CAUSE	TCPWM Counter Interrupt Cause Register	Determines the source of the combined interrupt signal
TCPWM_CNT_CTRL	Counter Control Register	Configures counter mode, encoding modes, one shot mode, switching, kill feature, dead time, clock prescaling, and counting direction
TCPWM_CNT_STATUS	Counter Status Register	Reads the direction of counting, dead time duration, and clock prescaling; checks if the counter is running
TCPWM_CNT_COUNTER	Count Register	Contains the 16-bit counter value
TCPWM_CNT_CC	Counter Compare/Capture Register	Captures the counter value or compares the value with counter value
TCPWM_CNT_CC_BUFF	Counter Buffered Compare/Capture Register	Buffer register for the counter CC register; switches period value
TCPWM_CNT_PERIOD	Counter Period Register	Contains upper value of the counter
TCPWM_CNT_PERIOD_BUFF	Counter Buffered Period Register	Buffer register for the counter period register; switches compare value
TCPWM_CNT_TR_CTRL0	Counter Trigger Control Register 0	Selects trigger for specific counter events
TCPWM_CNT_TR_CTRL1	Counter Trigger Control Register 1	Determine edge detection for specific counter input signals
TCPWM_CNT_TR_CTRL2	Counter Trigger Control Register 2	Controls counter output lines upon CC, OV, and UN conditions
TCPWM_CNT_INTR	Interrupt Request Register	Sets the register bit when TC or CC condition is detected
TCPWM_CNT_INTR_SET	Interrupt Set Request Register	Sets the corresponding bits in the interrupt request register
TCPWM_CNT_INTR_MASK	Interrupt Mask Register	Mask for the interrupt request register
TCPWM_CNT_INTR_MASKED	Interrupt Masked Request Register	Bitwise AND of the interrupt request and mask registers

Table 21-59. Peripheral Interconnect Trigger Group Control Registers

Register	Name	Description
PERI_TR_CTL	Trigger Control Register	This register provides software control over trigger activation.
PERI_TR_GROUP3_TR_OUT_CTLx	Trigger Control Register	This register specifies the input trigger for a specific output trigger in trigger group3.

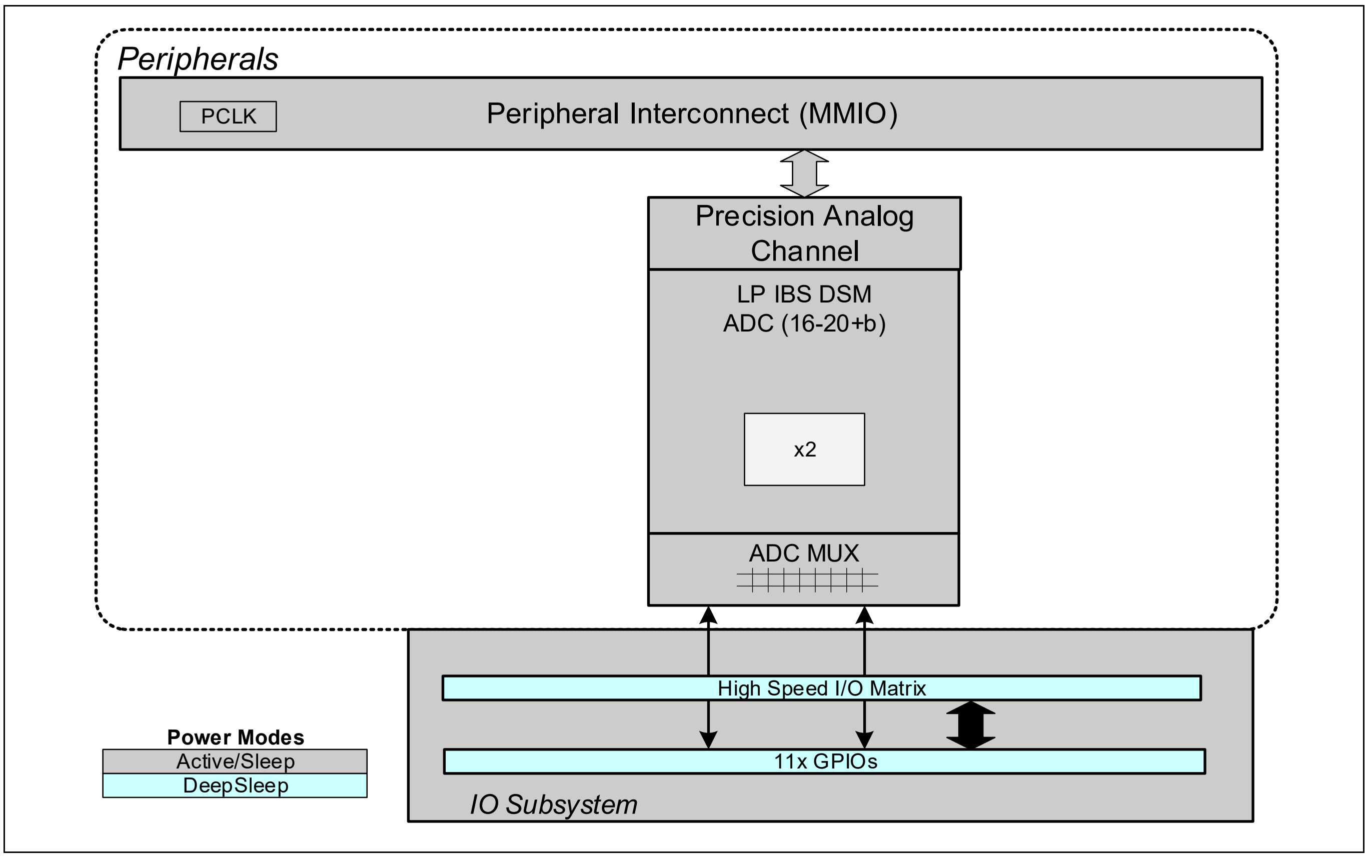
Section E: Analog System

This section encompasses the following chapter:

- **"Precision analog channel subsystem"** on page 276

Top Level Architecture

Analog System Block Diagram



22 Precision analog channel subsystem

The PSoC™ 4 HV PA precision analog channel subsystem (PACSS) is a high-performance data acquisition subsystem consisting of two physical channels. The PACSS contains the following blocks (see **Figure 22-1**):

- Analog delta-sigma modulator (DSM) system
  - Two analog channels
  - Channel multiplexer
- Digital Data System
  - Four digital channels (with/without FIR filter)
  - Data storages
- Automatic Gain Control
  - Gain multiplexer
- I/O Components
  - Input multiplexer
  - Two high-voltage (HV) input dividers[1)]
  - On-die temperature sensor
  - External temperature sensor

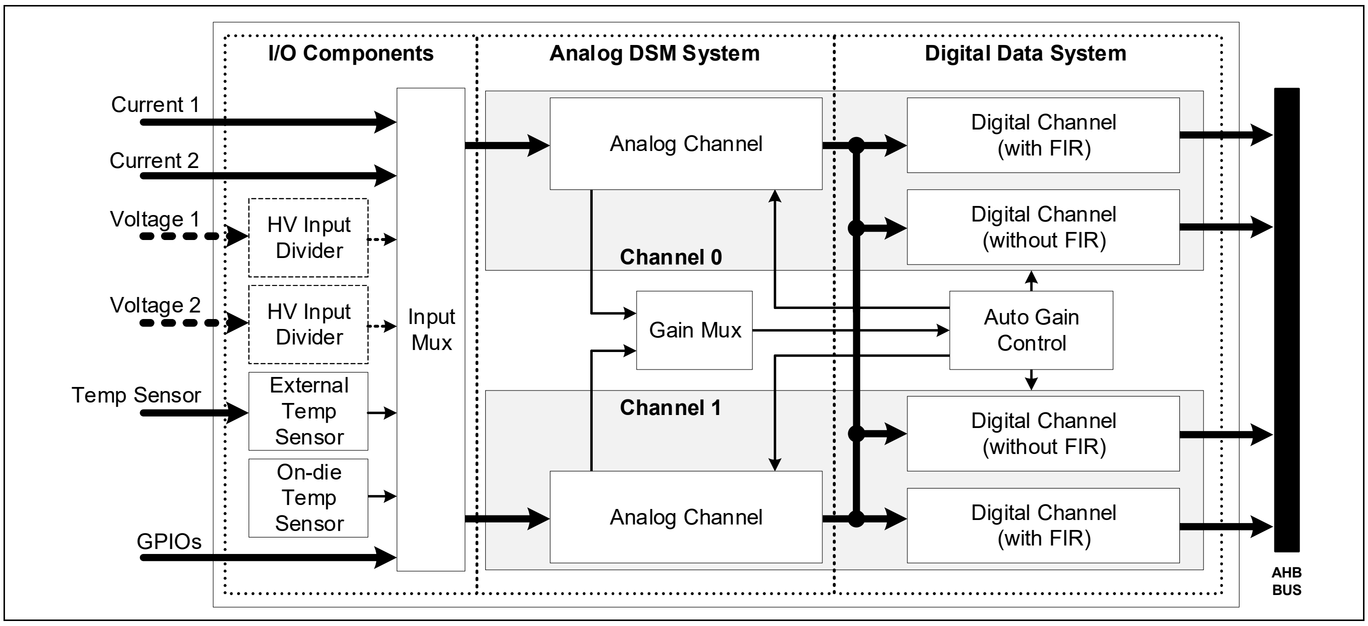


Figure 22-1. Block diagram

1) The HV input dividers are part of the High-Voltage Subsystem. See **"VDIVIDER (high-voltage divider)"** on page 357 for detailed information.

22.1 Features

- Two precision ΔΣ ADCs (16–20+ bits)
  - Maximum 48 ksps sample rate at OSR = 64
- Analog multiplexer selects from the following inputs:
  - Two current channels
  - Two high-voltage dividers
  - On-die temperature sensors
  - External NTC temperature sensor
  - GPIO pads
- Current channel with automatic gain
  - Enables measuring large starting currents or small battery-off currents
  - Gain range of 1 to 512, in powers of 2
  - 400 nVrms input referred noise at Gain = 512
  - ±500 nV offset at Gain = 512
  - Support for input voltages from –300 to +300 mV
  - Automatic or manual gain control
  - Optional chopping to minimize offset voltage error
- Voltage channel with HV input divider
  - Battery voltage to levels compatible with the ADCs by HV input divider
  - Static gain range of 0.5–512, in powers of 2
  - Nominal divider ratio is 24x (28.8 V full-scale)
  - Optional divider ratio is 16x (19.2 V full-scale)
  - Optional chopping to minimize offset voltage error
- Temperature and diagnostic channels
  - Supports on-die and external temperature sensing
  - On-die temperate sensing method is delta V_{BE}
- Up to four configurable digital channels
  - Filter and post-processing options
  - Offset and gain correction
- Scaler (9-bit left shift value)
- Decimator with programmable over-sampling ratio and order (per digital channel)
  - Sinc3 or Sinc4 filter, option is configurable
  - OSR = 2 to 128: (DR)
  - Sinc2 Filter (Rate Reducer)
  - OSR = 1 to 32 (DR2)
- Moving average: The last two or four results are averaged or summed
- Finite Impulse Response (FIR) filter
  - Configurable up to 64 taps
  - Each stage is 32 bits
  - 16-bit programmable coefficients
- Sequencer (per analog channel)
  - Supports simultaneous sampling
  - Incremental mode (16 ksps)
  - Continuous mode (48 ksps)
  - Digital channel scheduler
- Data storage
  - 32-bit ADC results per digital channel
  - 32-bit accumulated ADC results per digital channel

- Post processor
  - Range detection: 32-bit high and low thresholds
  - Interrupts: data valid, range detection, data saturated
  - Triggers: data valid
- Trigger function
  - Perform simultaneous measurements
  - One device uses a GPIO output to simultaneously trigger the on-chip ADC measurements while signaling other PSoC™ 4 HV PA devices to trigger measurements
  - Trigger signal can be set using either a timer or software to write to the GPIO

22.2 How it works

This section includes the following contents:

- PACSS measurement and acquisition system
- Introduction of each PACSS block

22.2.1 PACSS measurement and acquisition system

Figure 22-2 shows the block diagram of the PACSS measurement and acquisition system.

The analog DSM system is comprised of an input multiplexer, programmable gain amplifier (PGA), anti-alias filter (AAF), diagnostic multiplexer, buffer, and a delta-sigma modulator. The digital data system is comprised of a scaler, decimator, FIR-type digital filter, accumulator, comparator, and offset and gain calibrations.

Also included in this subsystem is the auto gain correction (AGC) circuit, and a temperature sensor with diagnostics capability. Each digital channel can quickly switch between input sources to create a "virtual" ADC by using the digital data system. This channel can be used for diagnostic purposes.

Sequencer is used to generate control signals for performing all the functions of the channel.

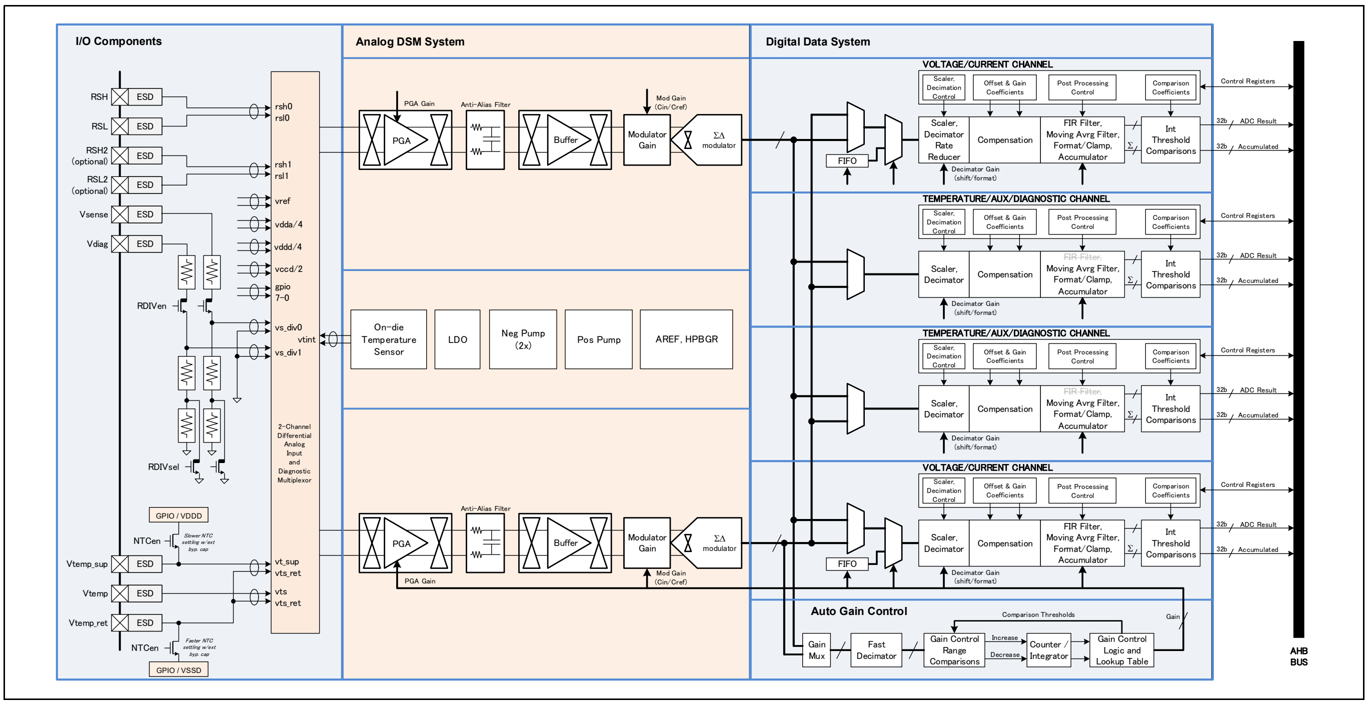


Figure 22-2. PACSS measurement and acquisition system

The PSoC™ 4 HV PA PACSS is divided into the following blocks:

- Analog DSM System
  - Sequencer
- Digital Data System
- Automatic Gain Control
- I/O Components

22.2.2 Analog DSM system

The PSoC™ 4 HV PA PACSS has two delta-sigma analog-to-digital converters (ADCs) to perform 16–20+ bit measurements at a sample rate of up to 48 ksps for continuous measurements. Higher resolution can be achieved at slower sample rates by accumulating more modulator results in the decimator.

The analog portion of the ADCs (see **Figure 22-3**) consists of a PGA, AAF, diagnostic multiplexer, buffer, and a delta-sigma modulator.

The analog DSM system receives a differential signal selected through the analog multiplexer. This differential signal is received by a programmable gain amplifier (PGA). The output of the PGA feeds a low-pass anti-alias filter (AAF) with a bandwidth of ~30 kHz. A buffer amplifier drives the DSM modulator – this amplifier has high bandwidth to settle the modulator capacitors to better than 16 bits each time they are settled. The modulator uses capacitor dividers to set gain. The modulator is a third order with switched capacitor amplifier circuits. The modulator produces a multi-level digital bitstream sent to the digital channel.

PGA and buffer amplifiers have offset trim, which reduces to offset these sub-blocks to <0.5 mV (across temperature). Offset trim is done during production. To reduce offset further, there is circuit chopping in PGA, buffer, and modulator. Chopping frequency is programmable from Fs/2 to Fs/256 (in 2's powers), where default is Fs/32 (PACSS_ACHANx_DPATH_CTL.BUF_PGA_FCHOP). Fs here is the modulator clock.

There is also a channel chop mode setting (PACSS_ACHANx_CHOP_CTL.CHOP_MODE):

- 00 - OFF: Channel Chopping and Buffer Cross Chopping Disabled
- 01 - Channel Chopping Enabled
- 11 - Buffer Cross Chopping Enabled
- 10 - Not supported

Channel chopping is between the inmux and the modulator output using the modbit.

Buffer cross chopping is on the input and output of the buffer using the modbit.

Both modes toggle the modbit when the PACSS_ACHANx_CHOP_CTL.SMP_CNT value is reached.

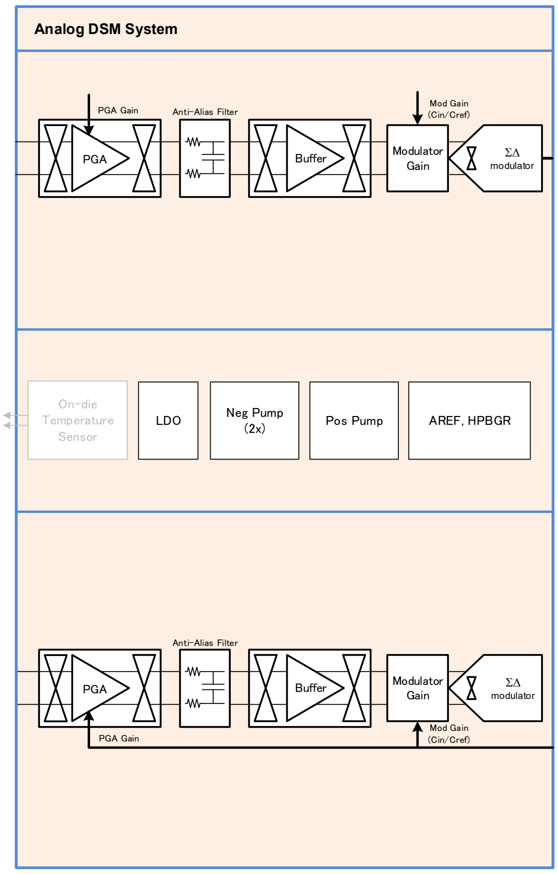


Figure 22-3. Analog DSM system block diagram

Figure 22-4 shows a simplified diagram of an analog channel. Analog multiplexers to bypass and swap blocks for diagnostics are omitted for clarity.

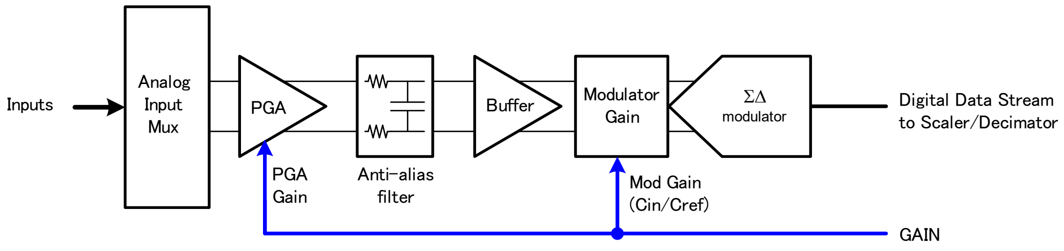


Figure 22-4. Analog channel - simplified diagram

Lowest input referred noise is achieved when PGA is set to its highest gain before feeding it to the DSM. FIR and Moving Average Filters in digital channels further reduce the noise by reducing bandwidth and averaging.

ADC measurements can be triggered by software or hardware. Hardware triggers can be generated by timers or GPIO inputs. The ADCs can be triggered independently or simultaneously. A trigger function can be implemented to allow multiple PSoC™ 4 HV PA devices to perform simultaneous measurements. One device uses a GPIO output to simultaneously trigger the on-chip ADC measurements while signaling other PSoC™ 4 HV PA devices to trigger measurements. The trigger signal can be set using either a timer or software to write to the GPIO.

22.2.2.1 Programmable gain amplifier

The PGA is a fully-differential pseudo-instrumentation amplifier implemented with two opamps. The PGA can be programmed for gains of 1 to 32 (PACSS_DCHANx_PGA_GAIN_CTL.GAIN). Additional channel gain can be obtained later in the channel with the DSM (typically modulator gain of 1 to 16 is used). Note that this gain setting is overridden by the AGC values if PACSS_MMIO_PACSS_CTL.AGC_EN = 1 and PACSS_DCHANx_DCHAN_CTL.AGC_GAIN_EN = 1. Therefore, the current value of the PGA gain cannot be read from this register during AGC mode.

The PGA gain can be programmed via the internal bus or comes from the AGC (PACSS_MMIO_STATUS.AGC_CURR_GLVL). This enables large dynamic range while maintaining good noise performance. The AGC selects the PGA gain to ensure that the output of the PGA is below the ADC reference. In this way over-ranging of the PGA. such as at high input signals can be avoided.

The PGA input range is –250 mV to 1.2 V (For RSH/RSL input pins, the input range is –300 mV to +300 mV). A negative pump is used to allow inputs to be below ground. This PGA has low-noise and input capacitance of about 1.5 pF. The PGA includes input/output chopping, which the sequencer can enable, and digital offset trim.

22.2.2.2 Anti-alias filter

The anti-alias filter (AAF) is a first order RC filter with a 30-kHz corner frequency. 30 kHz is selected to give 40 dB attenuation at modulator sampling frequency, which is 3.072 MHz. The filter also has bypass switches, which allow the filter to be removed from the signal path, and a set of fast switches that bypass the “R” in the RC filter, moving the corner frequency approximately 600 kHz. **Figure 22-5** shows the AAF. To alternately measure two signals with one channel, the anti-alias filter can be temporarily shorted since the AAF needs about 60 µs to settle to 16-bits before conversions can start - the measurement sample rate needs to slow to 4 ksps if the AAF is not bypassed.

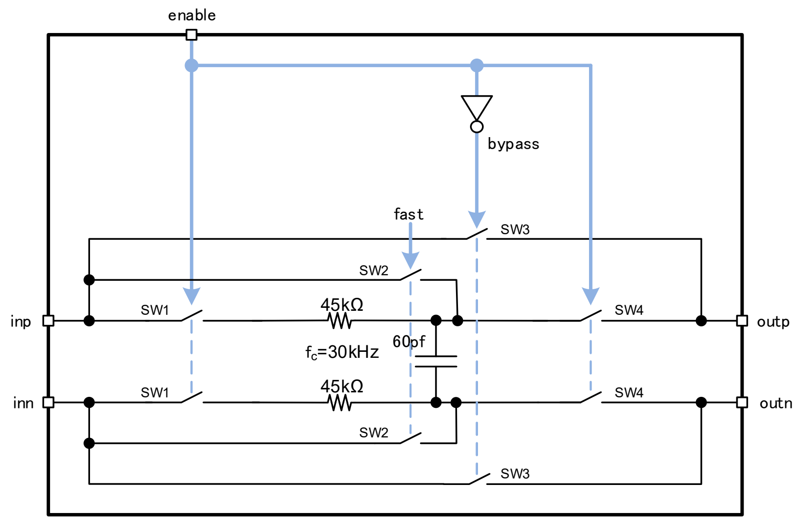


Figure 22-5. Anti-alias filter

22.2.2.3 Buffer amplifier

The buffer amplifier converts the high-impedance AAF filter output to low impedance for driving the DSM modulator. The modulator configuration used by in battery monitoring and management applications is typically 6.4 pF. The charge on the modulator capacitors must be restored every conversion cycle (3.072 MHz at maximum conversion rate), which requires a low-output impedance and high-unity bandwidth of about 18 MHz to allow the modulator capacitors charge to be restored and settle within a half clock cycle. The buffer includes digital offset trim and chopping, which can be enabled or disabled with a control signal.

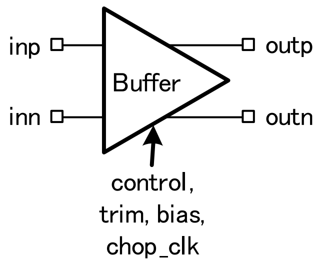


Figure 22-6. Buffer amplifier

22.2.2.4 Delta-sigma modulator

Figure 22-7 is a block diagram of a Delta-Sigma Modulator (DSM).

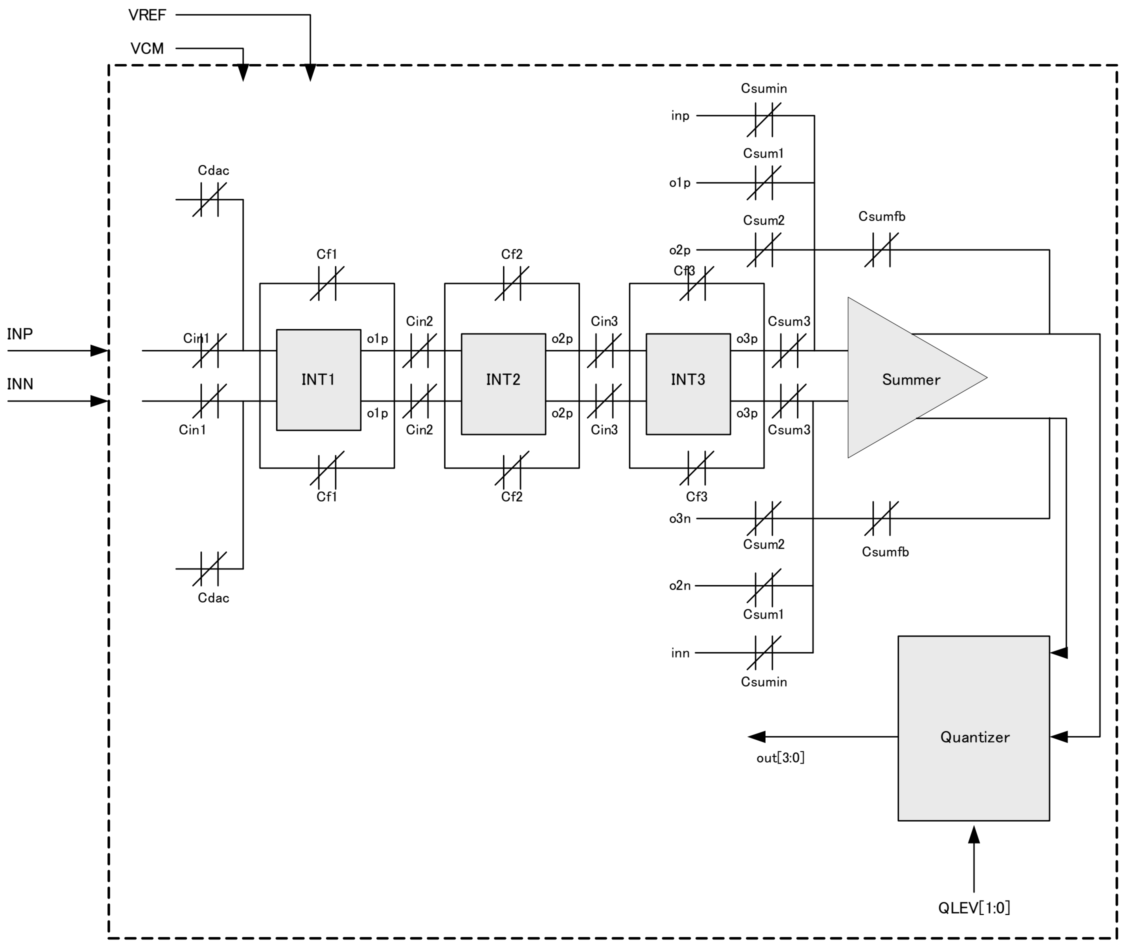


Figure 22-7. Delta-sigma modulator

The delta-sigma modulator works by taking the difference between input and feedback signals (delta) and accumulating that difference (sigma) to produce a digital output. The digital stream goes to a decimator, which converts the fast oversampled bit stream into slower high-resolution results.

This block is implemented with a switched capacitor architecture implementing a third order Cascade of Resonators Feed-forward architecture. The quantizer uses nine quantization levels.

The DSM has built-in overload detection with programmable detection threshold. The input common mode range is rail-to-rail (0 V to V_{DDA}, V_{DDA} = 2.7 V–3.6 V).

Modulator has typical attenuation of 15% to ensure the loop stability. Gain calibration of the Analog Channel corrects this attenuation (see **Table 22-9**).

Gain is programmable from 0.5x to 4x (with most battery monitoring and management applications using 1, 2, and 4, in powers of 2). Gain is selected by adjusting the size of the inputs caps, to reference gaps with more gain when larger input caps and smaller reference caps are selected. Programming of the capacitors is binary weighted with selectable resolution of 12 fF, 50 fF, or 100 fF.

A bi-directional data weighted averaging (DWA) based dynamic element matching scheme is used for capacitor mismatch cancellation. Register option exists to change Dynamic Element Matching Scheme (DEM) to be unidirectional instead of bi-directional.

Chopping is possible at programmable frequency in the first integrator to cancel the input path mismatch between buffer amplifier offset and 1/f noise.

Sign inversion support is controlled by external input to support applications such as full wave rectification and channel chopping.

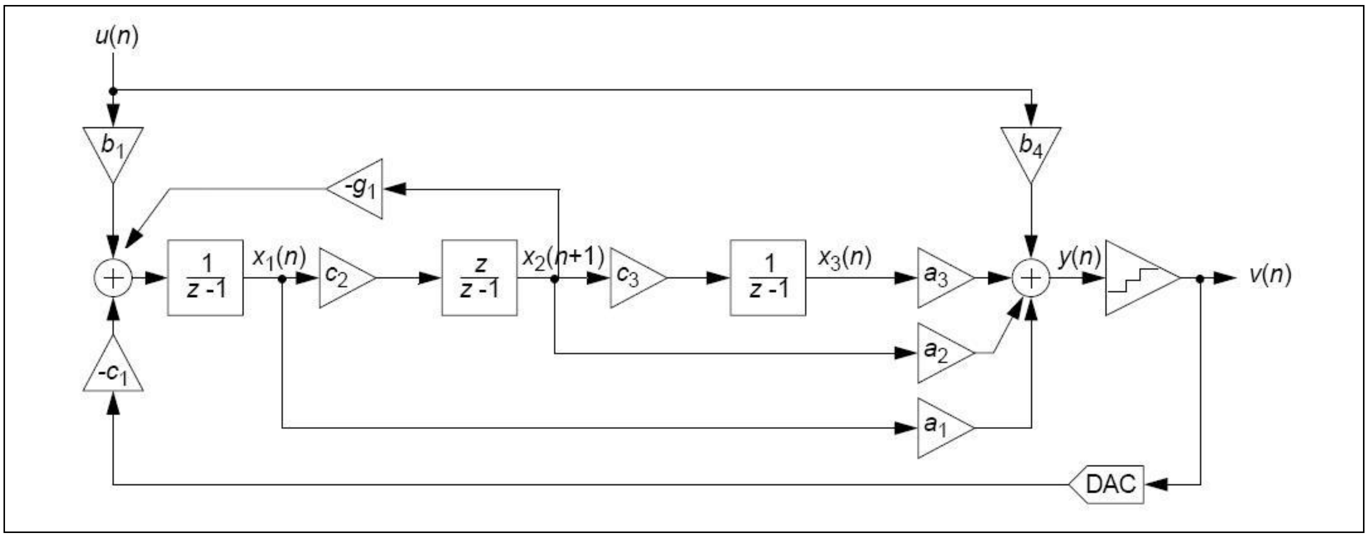


Figure 22-8. Delta-sigma modulator architecture

This modulator produces an 8-bit thermometric code, which is converted to 4-bit 2’s complement before scaling and decimation in the digital channel. The Cascade Resonant Feed-Forward modulator architecture is illustrated in **Figure 22-8**.

22.2.2.5 Dynamic element matching

The Delta-Sigma Modulator has a multi-bit quantizer; therefore, the reference DAC is multi-level. This causes linearity issues, which is solved by using dynamic element matching (DEM). Two types of DEM are supported. These are selected by PACSS_ACHANx_DEM_CTL.SCRAM0_EN and SCRAM1_EN bits and enabled by setting DEM_EN bit to high. DWA uses data weighted averaging technique for mismatch cancellation. The bi-directional DWA (Bi-DWA) alters the direction of pointers used for rotation in every cycle to avoid a repetition pattern of the DWA for low-value DC inputs. Alternate DWA (ADWA) is an alternate scheme where the pointer direction is reversed only on detection of repeating codes. The PACSS_ACHANx_DEM_CTL register is used to control DEM mode.

Table 22-1. DEM settings

SCRAM1_EN (EN_DWA)	SCRAM0_EN (EN_ADWA)	Function
0	0	Bi-DWA (default)
0	1	ADWA (not recommended)
1	0	DWA
1	1	Not used

22.2.2.6 Reference system

The main reference to modulator DAC comes from the High-Precision BandGap Reference (HPBGR). Other options are available for diagnostic features.

An external 470-nF (typ) capacitor is required for HPBGR reference to improve absolute accuracy and SNR by reducing noise. This capacitor is connected between V_{REFH} and V_{REFL} pins. V_{REFL} must not be shorted to ground at PCB level. V_{REFL} is Kelvin connected to V_{SSA} on the device.

The quantizer of the modulator uses the same reference as the DAC by default, but with a source-follower amplifier so that there is no constant load on HPBGR. Quantizer reference can also be V_{CCA} (1.8 V), but this will increase quantization noise (which will degrade SQNR). The default quantizer reference is HPBGR voltage (PACSS_ACHANx_REF_CTL.VREF_QTZ_SEL = 0).

The common mode for the modulator amplifiers is programmable. Available voltages are 0.7 V and 0.8 V from HPBGR and also V_{CCA} (1.8 V)/2. Default common mode is 0.8 V from HPBGR (PACSS_ACHANx_REF_CTL.VCM_SEL = 0).

The PACSS_DCHANx_SMP_REF_CTL register is used to control V_{REFH} and V_{REFL} connection. V_{REF} connections are recommended for each convert mode. **Table 22-2** shows the reference selection by mode.

Table 22-2. Reference selection by mode

Mode	Pos Ref	VREFH_SEL[2:0]	Neg Ref	VREFL_SEL[5:4]	Comments
Normal mode	VREFH Direct	0x00	VREFL	0x00	–
Internal temp sens	VREFH Direct	0x00	VREFL	0x00	–
External temp sens	VTS_REF	0x03	VTS_RET	0x03	–
VREFH measurement	VREF SRSS	0x02	VSSA_SRSS	0x01	VREFH is input to the channel
Other diagnostic	VREFH Direct	0x00	VREFL	0x00	Measuring power supplies, grounds, and so on

VREF measurement with a gain of 1x can measure the results even the ADC's reference is 1.2 V. It is because the DSM modulator attenuates the input by ~0.85x gain (see **Table 22-9**).

22.2.2.7 Negative pump

The negative pump produces –1.2 V, which is used by the programmable gain amplifiers and –0.8 V, which is used to bias the isolated p-well (body) of N-MOS switches that connect to signals which can go up to 300 mV below ground. The PACSS_ACHANx_PUMP_CTL register is used to control the negative pump.

The negative pump requires a 24-MHz clock. The clocks are created by HFCLK (clk_hf) or PUMP clock (clk_pump) from the SRSS (see the **"Clocking system"** on page 93 for more details).

Pump Clock Configuration

There is a functional option that will always save power without impacting performance by changing the clock configuration. The power can be saved by changing the following registers:

- PACSS_ACHANx_PUMP_CTL.CLOCK_SEL sets to 0x2 or 0x3

0x0: pump clock sourced by clk_pump/2, use if clk_pump is set to 48MHz

0x1: pump clock sourced by clk_pump/1, use if clk_pump is set to 24 MHz

0x2: pump clock sourced by clk_hf/2, use if clk_hf is set to 48 MHz

0x3: pump clock sourced by clk_hf/1, use if clk_hf is set to 24 MHz

- SRSSHV.CLK_SELECT.PUMP_SEL sets to 0x0

0x0 : GND : No clock, connect to gnd

0x1 : IMO : Use main IMO output

0x2 : HFCLK : Use clk_hf (using selected source after predivider but before prescaler)

Note: To save power, the pump clock can be disabled (PUMP_SEL=0x0) for normal operation, then re-enabled prior to calling FLASH operations. Each FLASH operation should confirm that pump clock is enabled.

22.2.2.8 Positive pump

The positive pump is included in the PACSS block as risk mitigation. Single positive pump is used for both channels. The PACSS_MMIO_VPOS_PUMP_CTL register is used to control the positive pump, and enabled via the PACSS_MMIO_PACSS_CTL.VPOS_PMP_EN bit. By default, the positive pump is disabled. In this mode, pump output is bypassed to VDDA.

22.2.2.9 Chopping configuration

Chopping is used to reduce offset and 1/f (Flicker) noise. Various chopping schemes are implemented in the PACSS. See **Table 22-3** for default values and **"Sequencer timing"** on page 315 for each channel setting.

Table 22-3. Default chopping settings

Register	Field	Comment	D/T Channel	V Channel	I Channel
PACSS_ACHANx_CHOP_CTL	SMP_CNT	Chopping Sample Count number of samples to count before toggling modbit	N/A	N/A	3
	AAF_SHORT_R_CNT	Anti-Aliasing Filter Short Resistor Count number of DSM clock cycles	N/A	N/A	N/A
	DEC_BLANK_CNT	Decimator Blanking Count number of DSM clock cycles	N/A	N/A	N/A
	CHOP_MODE	Set channel/buffer cross chopping mode	0	0	3
	CIRCUIT_CHOP	Disconnect PGA-AAF for 0-3 clk cycles at PGA chopping rate	2	2	2
	CIRCUIT_2ND_EN	Second Stage Circuit Chopping Enable	1	1	1
	CHOP_RST_EN	Reserved (no functionality)	0	0	0
PACSS_ACHANx_PGA_CTL	PGA_CHOP_EN	Chopping Enable for PGA	1	1	1
PACSS_ACHANx_BUF_CTL	BUF_CHOP_EN	Chopping Enable for Buffer	1	1	1
PACSS_ACHANx_MOD_CTL	MOD_FCHOP	Modulator Chopping Clock Frequency Selection	Fs/32	Fs/32	Fs/32
	MOD_CHOP_EN	Modulator Chopping Enable	1	1	1
PACSS_ACHANx_DPATH_CTL	BUF_PGA_FCHOP	Chopping Clock Frequency selection for Buffer and PGA	Fs/32	Fs/32	Fs/32
	BUF_PGA_CHOP_CLK_EN	Buffer and PGA Chopping Clock Enable	1	1	1
PACSS_MMIO_HPBGR_CTL	CHOP_EN	HPBGR Chopping Enable	1	1	1
	HPBGR_FCHOP	HPBGR Chopping Clock Frequency selection	Fs/4	Fs/4	Fs/4

22.2.2.10 Power configuration

Power control registers save power in negative pump, modulator, PGA, and buffer. **Table 22-4** shows the recommended settings for each register.

Table 22-4. Recommended power and clock settings

Register	Field	Comment	Value	Power Level
PACSS_ACHANx_PUMP_CTL	VNEG_PWR_MODE	Negative pump power mode	3	–
	CLOCK_SEL	Negative pump clock selection	2 or 3	clk_hf/2 or clk_hf/1
PACSS_ACHANx_MOD_CTL	POWER1	First stage opamp power level control	5	88%
	POWER2_3	The power control for the second and third integrator stages	3	100%
	POWER_COMP	The power control for the quantizer block	2	100%
	POWER_SUM	The power control for the summer block	3	100%
PACSS_ACHANx_BUF_CTL	BUF_PWR_LEVELS	Buffer power levels	2	78%
PACSS_ACHANx_PGA_CTL	PGA_PWR_LEVELS	PGA power levels	1	58%
SRSSHV.CLK_SELECT	PUMP_SEL	Clock source for charge pump clock	0[a)]	No clock

a) Should re-enable prior to calling FLASH operations. Each FLASH operation should confirm that pump clock is enabled.

22.2.2.11 Modulator data path

This section describes the flow of the modulator data through the design. **Figure 22-9** shows the block diagram of the modulator data path.

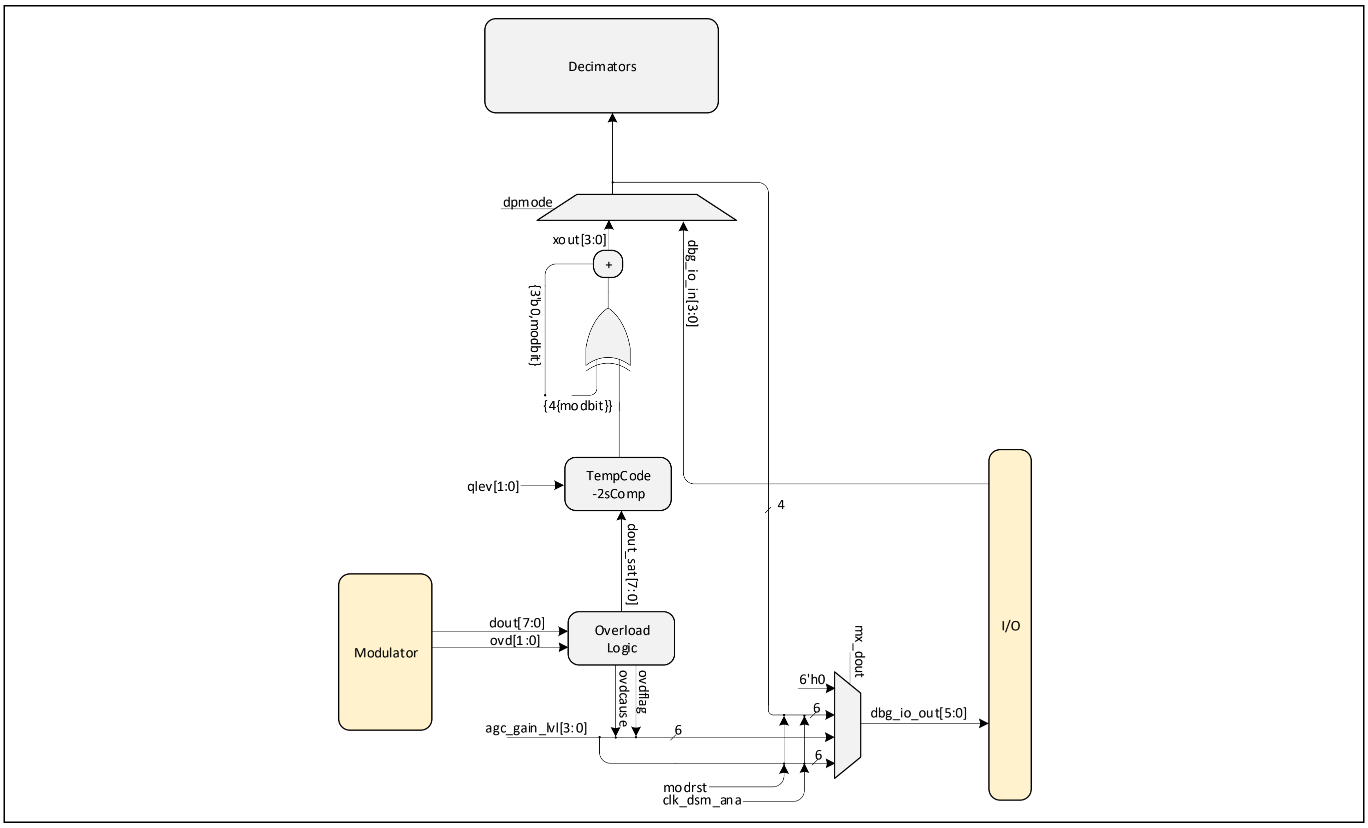


Figure 22-9. Modulator data path

22.2.2.12 Overload logic

The modulator outputs dout[7:0] and ovd[1:0] are routed to the analog channels data path logic. ovd[1:0] is the overload detect status bits: ovd[1] and ovd[0]. The output ovd[1] is set when an overload of ones is detected. The output ovd[0] is set when an overload of zeros is detected. The ovdflag signal is an output of the overload logic and is set if either the one or zero overload bit is set. The ovdcause bit is set for overload one detected and cleared for overload zero detected. The overload logic is enabled and configured with the PACSS_ACHANx_DPATH_CTL register.

When ovdflag is set it modifies the data to all zeros or all ones depending on the ovdcause bit before the two's complement conversion.

If the number of 1s or 0s at the modulator output exceed certain number (set in PACSS_ACHANx_DPATH_CTL.ODET_TH), then first, second, or third integrator stage feedback can be reset. RESET1_EN, RESET2_EN, and RESET3_EN fields are set to enable which capacitors are to be reset in the case of modulator overload. The RESET*_EN bits are used as a level of correction; in most cases start with setting RESET3_EN, then if more correction is required set RESET2_EN, and then RESET1_EN if needed.

Overload detect runs continuously, even when a conversion is not in progress. Overload detect may trigger even when a conversion is not in progress. To stop the modulator from running continuously, disable HPBGR chopping (PACSS_MMIO_HPBGR_CTL.CHOP_EN) and the free-running clock (PACSS_ACHAN0_DPATH_CTL.FCLOCK_EN).

Note that getting the overload interrupt (PACSS_DCHANx_INTR.OVERLOAD_INTR) indicates that the sample can be corrupted by the overload reset event. Getting multiple interrupts indicate that the block is configured incorrectly or there is an error with the signals being measured.

22.2.2.13 Quantization level

The two's complement conversion logic inputs the output data from the overload logic and a 2-bit quantization level configurable input (PACSS_ACHANx_DPATH_CTL.QLEV). The default value of quantization level is 9-level modulation.

Table 22-5. Two's complement conversion (QLEV)

Input		**Output**	
QLEV[1:0]	**dout[7:0]**	**dout2scomp[3:0]**	
2'b00	8'b00000000	4'b1111	–1
2'b00	8'b11111111	4'b0001	1
2'b01	8'b00000000	4'b1111	–1
2'b01	8'b00001111	4'b0000	0
2'b01	8'b11111111	4'b0001	1
2'b10	8'b00000000	4'b1100	–4
2'b10	8'b00000001	4'b1101	–3
2'b10	8'b00000011	4'b1110	–2
2'b10	8'b00000111	4'b1111	–1
2'b10	8'b00001111	4'b0000	0
2'b10	8'b00011111	4'b0001	1
2'b10	8'b00111111	4'b0010	2
2'b10	8'b01111111	4'b0011	3
2'b10	8'b11111111	4'b0100	4

22.2.2.14 Data path multiplexing

After the two's complement conversion, if channel chopping is enabled and the modbit is high, the data is inverted to create the signal xout[3:0]. In the default case xout is used as the decimator's inputs, but there is an option to route off-chip data directly to the decimator via dbg_io. This mux selection is controlled by PACSS_ACHANx_DPATH_CTL.MX_DIN.

The dbg_io port is a bi-directional port that can also be used to send the modulator output, modulator reset and clock, overload status bits, and the AGC gain level off-chip for debug purposes. This mux select is set by PACSS_ACHANx_DPATH_CTL.MX_DOUT.

The PACSS can have up to two analog channels, but there is only one dbg_io port per PACSS. If both channels attempt to use the dbg_io port, channel 0 will get priority.

22.2.3 Digital data system

The digital data system has four digital channels that include scaler, decimator, rate control, FIR filter, accumulator, comparator, offset, and gain calibrations. The digital channels process outputs from either of the two analog channels. Two channels are with FIR filter and the others are without the FIR Filter. The digital channel without FIR filter has every other feature.

The channels are typically used for current, voltage, temperature, and diagnostic measurements although they can be associated with any inputs. The digital system of the PACSS is largely autonomous; it performs acquisitions, filtering, post-processing, and data storage without firmware intervention. This allows real-time measurements without loading the CPU. The digital data system optionally generates various interrupts to enable further processing of the results, or to handle diagnosis errors. Triggers are also generated by the digital data system for system peripheral use cases.

Figure 22-10 shows a diagram of the digital data system. The digital data system is configured, then initiated with a trigger, which must be associated with one or both analog channels. At this point the sequencer of the analog channel determines which digital channels have selected it. It may be one, or up to all four digital channels. The sequencer loads the configuration from the first digital channel then starts the acquisition. The sample is converted, optionally filtered, optionally post-processed then stored. The sequencer will then move on to the next digital channel, if any, and repeat until it has converted all the digital channels linked to the sequencer. There are interrupts, status bits, and output triggers that the software and peripherals can use to track the events of the digital data system.

The PACSS register configuration is divided into three sections:

- Global configuration (pacss_mmio)
- Digital channel (dchan)
- Analog channel configuration (achan)

The global configuration (pacss_mmio) applies to the entire system. This contains the configuration for the sequencers, the temperature sensor, the AGC, and the logic that muxes data between dchan and achan slices.

The digital channel (dchan) contains the sample configuration data that will be sequenced, such as the pin to be sampled, and the filtering and post processing options. Each dchan slice contains a decimator, FIR filter (optional), a post-processing block, and data storage. There can be up to four dchan slices per PACSS. The default configuration of the digital data system will instance two dchans with a FIR filter and two dchans without FIR filter.

The analog channel configuration (achan) is the analog configuration that is static while the sequencer traverses between digital channels. Each achan contains the triggering logic, a sequencer, and data path (DPATH) logic.

In general, all register configuration must be set before triggering and static while conversions are running. The exceptions are noted in the register map.

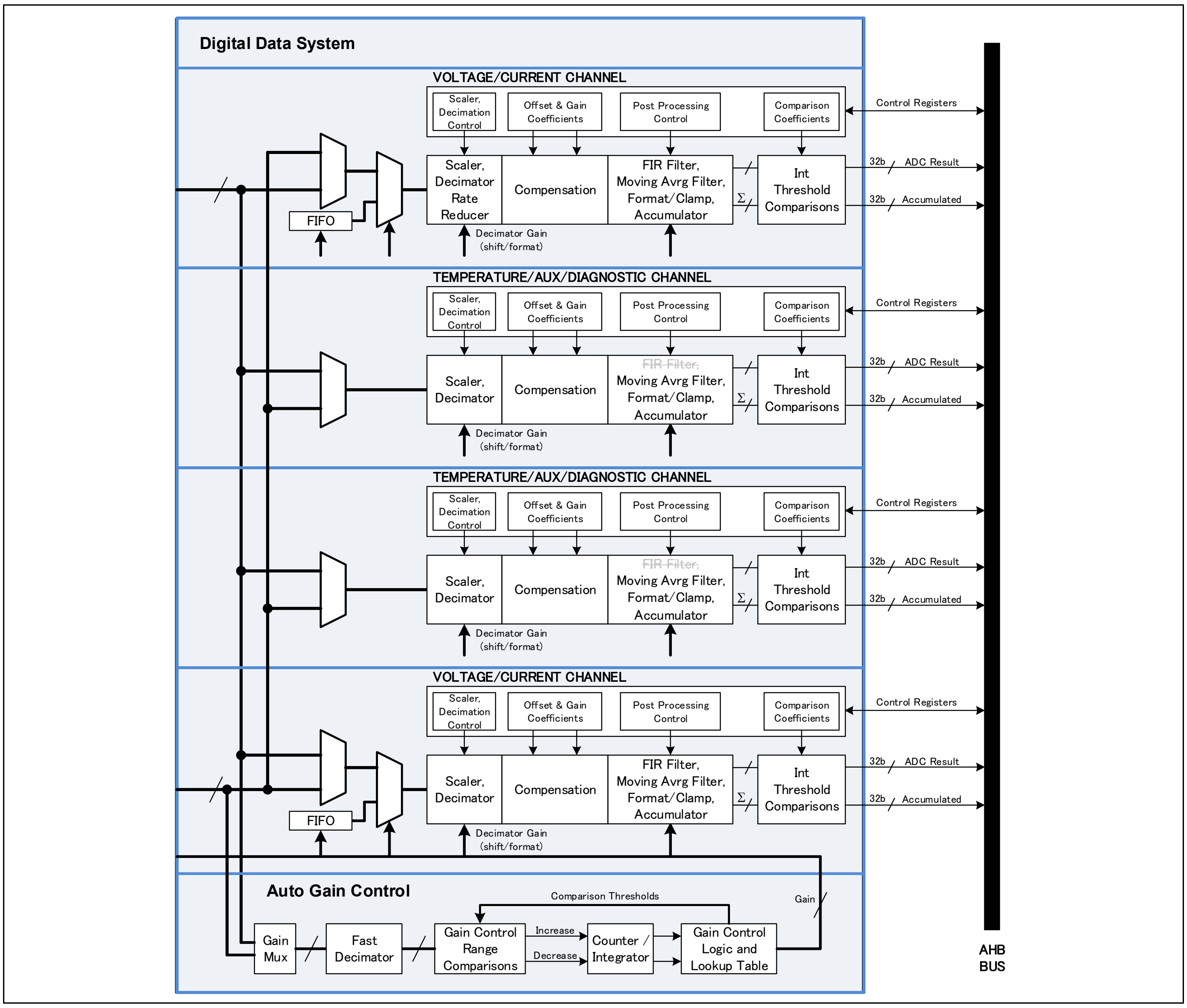

Digital Data System
VOLTAGE/CURRENT CHANNEL
TEMPERATURE/AUX/DIAGNOSTIC CHANNEL
Scaler, Decimation Control
Offset & Gain Coefficients
Post Processing Control
Comparison Coefficients
Scaler, Decimator Rate Reducer
Scaler, Decimator
Compensation
FIR Filter, Moving Avrg Filter, Format/Clamp, Accumulator
~~FIR Filter,~~
Int Threshold Comparisons
FIFO
Decimator Gain (shift/format)
Control Registers
32b ADC Result
32b Accumulated
Σ
Auto Gain Control
Gain Mux
Fast Decimator
Gain Control Range Comparisons
Increase
Decrease
Counter / Integrator
Gain Control Logic and Lookup Table
Comparison Thresholds
Gain
AHB BUS

Figure 22-10. Digital data system block diagram

The digital channel converts the modulator output data and includes scaling, filtering, and compensation. It also compares ADC values with thresholds to generate interrupts when thresholds are exceeded. **Figure 22-11** shows a simplified diagram of the digital channel path.

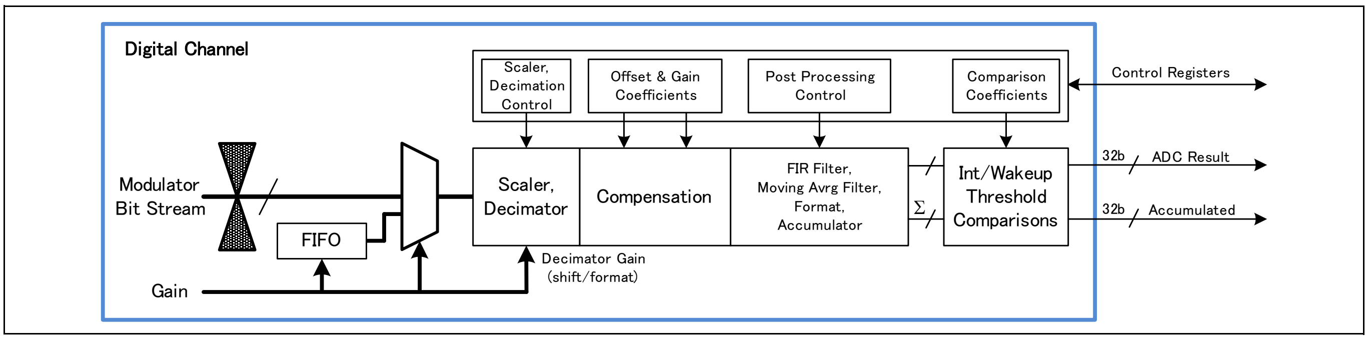


Figure 22-11. Simplified diagram of digital channel

The modulator bit stream first goes to a channel chopper, which can multiply the bit stream by +1 or –1. The voltage channel uses programmable fixed gain while the current channel can use either fixed or automatic gain.

A scaler is used so the LSB of the ADC can have the same weight regardless of gain. The LSB is established by the ADC resolution at maximum gain - for the current channel, the LSB is 0.715 mA with a gain of 512. The scaler multiplies the output of the modulator by 512/Gain to normalize results and maintain 0.715 mA for the LSB regardless of gain setting. To multiply by 2, the scaler shifts results up one bit. To cover a gain range of 1 to 512 means results can be shifted up to 9 bits.

There are two decimators – one is configurable as sinc3 or sinc4, and the other is used as rate reducer and is a sinc2.

The first decimator can be configured for third or fourth order operation. The decimation rate (DR) is programmable and can range from 2 to 128. Both the third and fourth order decimator configurations can be used for incremental or continuous mode.

The second decimator (rate reducer) DR2 is programmable and can range from 1 to 32. For instance, with a 3.072-MHz clock, DR is set to 64, and DR2 is set to 6 to achieve an 8 ksps sample rate.

The output of the decimator goes to a compensation block, which multiplies results for gain adjustment and adds constants for offset correction. Up to 64 tap finite impulse response (FIR) filter with programmable coefficients follows the compensation block. Results are then normalized to remove unused bits and averaging, accumulation, and threshold detection can be performed. Threshold comparison uses a window comparator, which can be programmed for high and low thresholds that trigger interrupts. Channel control registers are programmed by the CPU using the AHB bus. Results can be transferred by DMA or CPU. An interrupt can initiate a DMA transfer or notify the CPU data is available. The CPU can also poll for end-of-conversion to determine data is available.

Figure 22-12 shows the block diagram of the digital channel (DCHAN) data path.

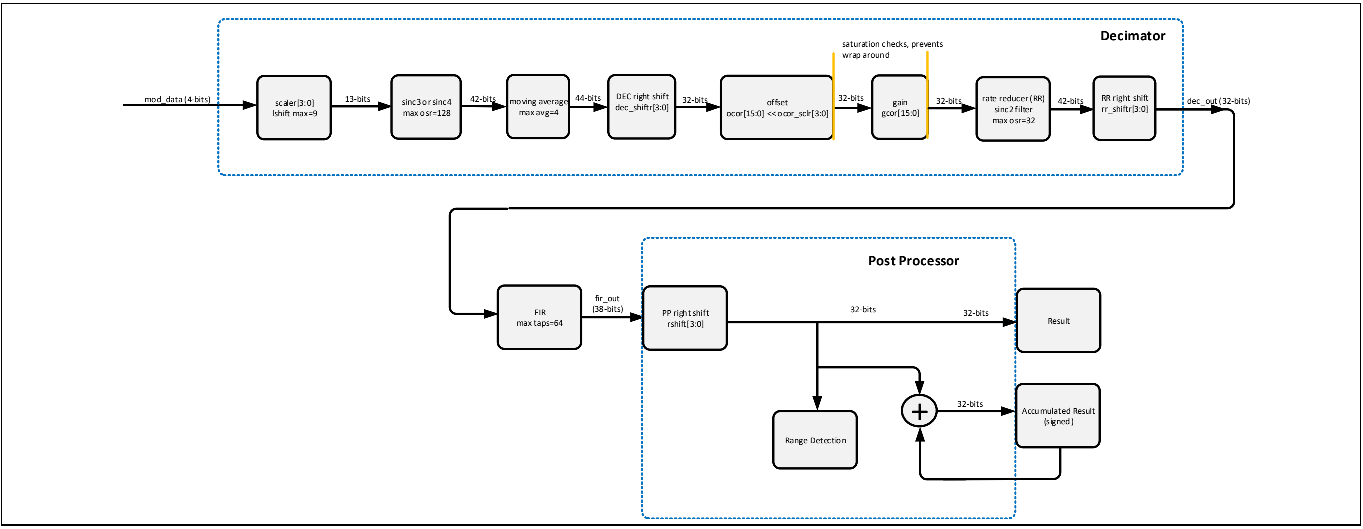

Decimator
saturation checks, prevents wrap around
mod_data (4-bits)
scaler[3:0] lshift max=9
13-bits
sinc3 or sinc4 max osr=128
42-bits
moving average max avg=4
44-bits
DEC right shift dec_shiftr[3:0]
32-bits
offset ocor[15:0] << ocor_sclr[3:0]
32-bits
gain gcor[15:0]
32-bits
rate reducer (RR) sinc2 filter max osr=32
42-bits
RR right shift rr_shiftr[3:0]
dec_out (32-bits)
Post Processor
FIR max taps=64
fir_out (38-bits)
PP right shift rshift[3:0]
32-bits
32-bits
Result
Range Detection
32-bits
Accumulated Result (signed)

Figure 22-12. Digital channel data path

22.2.3.1 Decimator

The decimator is either a three- or four-stage CIC filter, which consists of cascaded integrators followed by differentiators, implementing a comb filter. The number of stages is determined by the register bit PACSS_DCHANx_DEC_CTL.SINC_MODE. The decimator is clocked by clk_dsm, which runs at the same frequency as the modulator. The differentiators however are clock-enabled at a slower rate determined by the decimation rate. The decimation rate (DR) is configurable in the PACSS_DCHANx_DEC_CTL register to any integer value between 2 and 128. There are some DR value restrictions based on configuration choices, which are noted in *PSoC™ 4 HV PA Registers TRM*. Therefore, the differentiators are effectively clocked at Fs/DR.

Figure 22-13 shows a block diagram of the decimator.

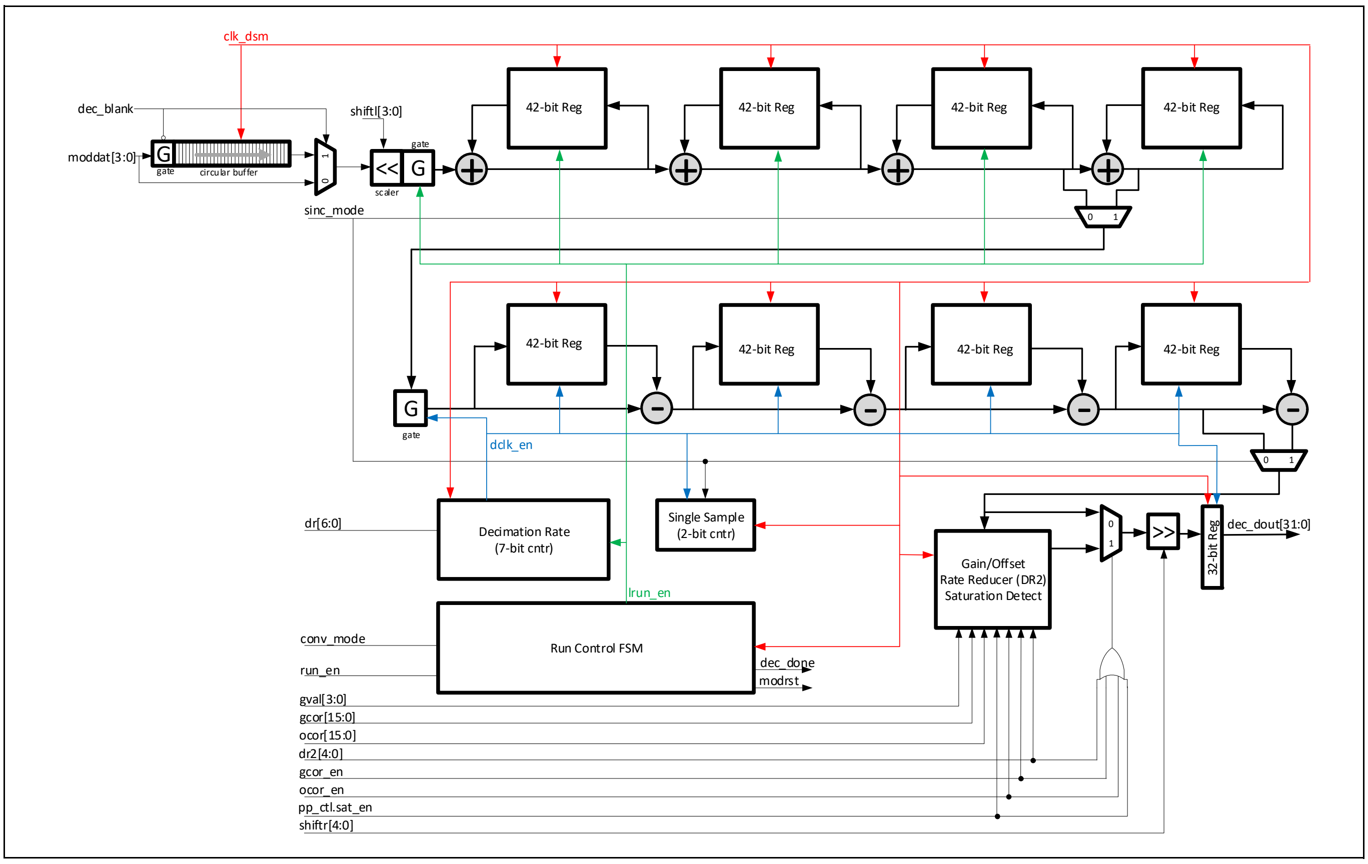


Figure 22-13. Decimator block diagram

The 7-bit decimation counter is used to produce a clock enable strobe for the comb (subtractor) section - allowing it to effectively clock at a rate slower than the sample rate. This counter divides Fs by a programmable value between 2 and 128. Depending on the configuration the filter is in, this full range may not produce valid results. The expectation is decimation ratios (DR) of 8 to 128.

The Single Sample counter is used to measure out the decimation periods (3 or 4) that constitute a single sample. In incremental mode, the filter needs clocks equal to (Fs) S x DR to produce a valid result. S = 3 when PACSS_DCHANx_DEC_CTL.SINC_MODE = 0, and 4 when SINC_MODE = 1. This counter is also used in continuous mode. When the modulator is reset, the first S-1 decimations cycles do not produce a valid data – thus they are not used as results.

A second sinc2 filter is used as a rate reducer. It is nothing more than an accumulator of the samples from the CIC filter. This uses the decimation ratio called DR2; it has a maximum value of 32.

With the 3.072-MHz clock (clk_dsm), output data rate is programmable between 0.75 to 48 ksps based on the DR values.

22.2.3.2 Circular buffer

The circular buffer is used to blank the decimator with recent but buffered modulator data while the analog settles after a gain or input mux selection change.

The decimators circular buffer can be activated by the sequencer or the AGC. When it is not activated, it continuously stores modulator data. The buffer holds 32 4-bit samples before it overlaps. When the circular buffer is activated, it no longer receives new samples and outputs the oldest data in buffer into the accumulation stage of the decimator.

22.2.3.3 Conversion modes

The supported conversion modes are incremental (single sample) and continuous. The conversion modes are set in PACSS_DCHANx_DEC_CTL.CONV_MODE register (0 - Incremental (Single Sample), 1 - Continuous).

Note that the primary (I and V) measurements should be made using continuous mode; performance is better in this mode. Incremental mode is meant for secondary (diagnostic) measurements.

When the Decimator Sinc mode is set in the PACSS_DCHANx_DEC_CTL.SINC_MODE register, the initial conversion time will change as shown in **Figure 22-14** and **Figure 22-15**.

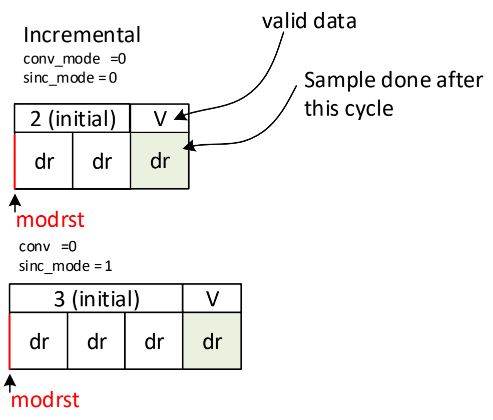


Figure 22-14. Incremental conversion (Single Sample)

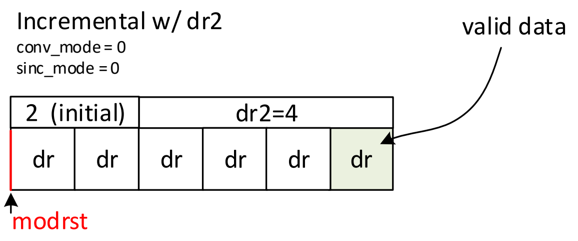


Figure 22-15. Incremental conversion w/ dr2 = 4

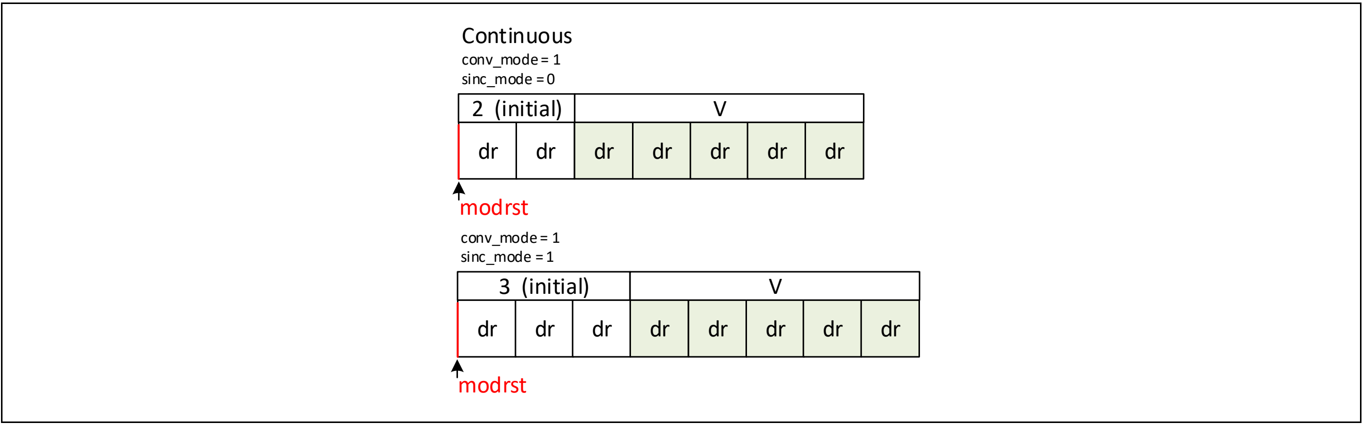


Figure 22-16. Continuous conversion

22.2.3.4 Left shift

The SHIFTL[3:0] fields are used to align the output of the CIC filter to a consistent location. SHIFTL depends on the input levels on mod data, decimation ratio, and whether the channel uses moving-sum averaging.

It can be used as a scalar to normalize the result according to gain applied by the PGA and modulator.

SHIFTL defaults to LSB aligned and all shifts are to the left (toward MSB). The register is programmed with PACSS_DCHANx_DEC_CTL.SHIFTL when AGC is not used. When AGC is used, each gain level has its own SHIFTL, found in PACSS_MMIO_GAIN_CFGx.SHIFT1. Frequently, this document will refer to SHIFTL; SHIFT1 may be substituted for configs which use AGC, where appropriate.

22.2.3.5 Right shift

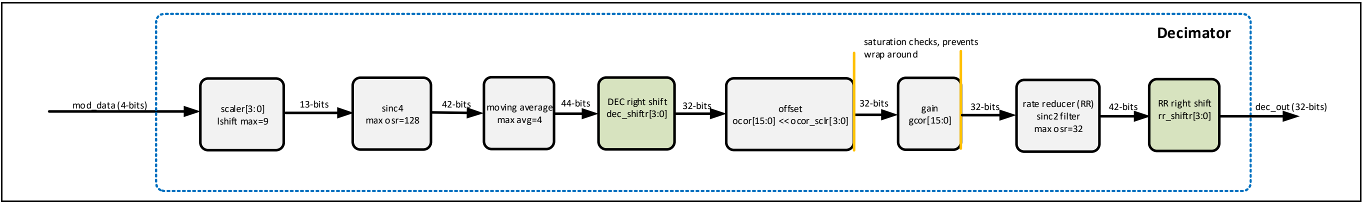


Figure 22-17. Decimator right shifting stages

The right shifts have two purposes:

- To work with the left shift to align the data in the datapath.
- To reduce the magnitude of the data so that it fits within the datapath limits.

The digital datapath has two right-shifting stages. The first stage is after the moving average computation, just before offset-correction is applied. The second right shift is after the rate-reducer.

The input of the offset-correction will truncate the most significant bits to reduce the bit-width to 32-bits. If the converted value surpasses 32-bits, this would result in a loss of MSB data. The data must be right shifted at the first stage in order to fit. Further, the gain profiles were developed at specific magnitudes, and right shifting is an easy way to ensure the data magnitude aligns with the gain profile. See **"Offset correction values alignment"** on page 298.

Similarly, the rate-reducer can output data that is too wide. The second right shifter can realign this data.

The amount to right shift at the first stage depends on the quantization level, decimator oversample ratios, number of decimator filter stages (sync order), and left shift scaler values.

Use the following equation to determine the right shift value for PACSS_DCHANx_DEC_CTL.SHIFTR:

$$\mathrm{DEC\_SHIFTR} = [\log2(\mathrm{qLev}) + 1] + \max(\mathrm{SHIFTL}) + \mathrm{N} \times \log2(\mathrm{DR}_1) + \log2(\mathrm{U}) - 32$$

qLev = Quantization level of the modulator. Usually 9.

max(SHIFTL) = Largest scaler shift used by any gain level.

N = Filter order of the first stage. Usually 3.

DR_1 = Oversample ratio of the first stage decimator.

U = Number of samples in the moving average. Often 4 or 1.

The next shifting stage is after the sync2 rate reducer. The rate reducer's output max width is 42 bits, but the adc result must be 32-bit. The 10 most significant bits of data would be truncated, so it is important that the second right shift reduce the data width to 32-bit.

Use the following equation to calculate PACSS_DCHANx_DEC_CTL.RR_SHIFTR:

$$\mathrm{RR\_SHIFR} = [\log2(\mathrm{qLev}) + \max(\mathrm{SHIFTL}) + \mathrm{N} \times \log2(\mathrm{DR}_1) + \log2(\mathrm{U}) - \mathrm{DEC\_SHIFR} + 2 \times \log2(\mathrm{DR}_2) + 1] - 32$$

A RR_SHIFTR value will guarantee that the number is not truncated during the PACSS processing. However, the magnitude of the output code still needs to be aligned with the input signal. See **"Decimator result calculation"** on page 297.

22.2.3.6 Moving average and sum

The moving average logic uses up to four 42-bit registers to store the last two or four samples and continuously compute the average or sum of these samples based on the configuration. The latency of the moving average logic is two clk_dsm cycles, regardless if average mode is set to 2 or 4. The configuration for moving average is contained in the PACSS_DCHANx_SMP_CTL register.

22.2.3.7 Offset and gain correction

Offset and gain correction are enabled via the PACSS_DCHANx_DEC_CTL register (OCOR_EN and GCOR_EN bits). If offset correction is enabled, the 16-bit PACSS_DCHANx_OFST_COR.OCOR value is left shifted by the PACSS_DCHANx_OFST_COR.OCOR_SCLR value then added to the decimation result.

If gain correction is enabled, the decimated result is multiplied by the 16-bit PACSS_DCHANx_GAIN_COR.GCOR value. The 4-bit PACSS_DCHANx_GAIN_COR.GVAL field is used to determine how many bits of the GCOR value are valid in the calculation. See the register description for more information. If both offset and gain correction are enabled the offset calculation is done before the gain multiplication.

If Auto Gain Control is enabled, the offset and gain correction are taken from PACSS_MMIO_OFST_COR1 and PACSS_MMIO_GAIN_COR1 registers. The OCOR_EN, OCOR_SCLR, GCOR_EN, and GVAL values are taken from each PACSS_DCHAN register settings as shown here.

Table 22-6. Details

Values	Description		Register name
OCOR_EN	Offset Correction Enable	←	PACSS_DCHANx_DEC_CTL.OCOR_EN
OCOR_SCLR	Decimator Offset Correction Coefficient Scaler	←	PACSS_DCHANx_OFST_COR.OCOR_SCLR
GCOR_EN	Gain Correction Enable	←	PACSS_DCHANx_DEC_CTL.GCOR_EN
GVAL	Number of valid bits minus one in Gain Coefficient registers	←	PACSS_DCHANx_GAIN_COR.GVAL

See **"PACSS calibration"** on page 330 for more detailed information.

22.2.3.8 Saturation

The decimator implements saturation logic that is used to prevent over- and under-flow wrap-around in the accumulator. When saturation is enabled the ALU in the decimator will not wrap if the most positive or negative number is exceeded. This feature is used to prevent wrap-around if the decimator is incorrectly configured for the application and produces a result larger than the accumulators can hold.

22.2.3.9 Decimator result calculation

The ADC's analog input is gained by a programmable gain amplifier (PGA) and by the modulator's capacitor settings, then it is compared to a reference voltage to produce a 4-bit output stream. This stream is left shifted, decimated, averaged, right shifted, offset/gain corrected, decimated again, and right shifted again.

To convert the decimator results to a voltage, use:

$$V_{out} = V_{range} \times \left(\frac{Co_{meas}}{Co_{max}}\right)$$

Co_{meas} = Counts of the measurement.

V_{range} and Co_{max} are constants which can be derived in advance.

$$V_{range} = \frac{V_{ref}}{G_{PGA} \times G_{cap}}$$

V_{ref} = Reference voltage. For example, 1.2 V.

G_{PGA} = Gain of the programmable gain amplifier. For example, 1.

G_{cap} = Gain introduced by the modulator's capacitor arrays. For example, 1/2.

$$Co_{max} = M \times U \times 2^{S_L + N_1 \log2(DR_1) - S_R + N_2 \log2(DR_2) - S_{RR}}$$

M = Maximum modulator output. For example, 4.

U = Contribution from moving-sum averaging. For example, 4.

S_L = SHIFTL value. For example, 8.

N_1 and DR_1 are the first decimator stage's filter order and decimation ratio. For example, 3 and 128.

S_R = DEC_SHIFTR value. For example, 3.

N_2 and DR_2 are the second decimator stage's filter order and decimation ratio. For example, 2 and 12.

S_{RR} = RRSHIFTR value. For example, 7.

Using the example values:

$$V_{range} = \frac{1.2V}{1 \times \frac{1}{2}} = 2.4V$$

$$Co_{max} = 4 \times 4 \times 2^{8 + 3\log2(128) - 3 + 2\log2(12) - 7}$$

$$= 0x4800\_0000$$

The LSB (V) can be found by assuming Co_{meas} = 1.

$$V_{LSB} = V_{out}(1) = 2.4 \times \frac{1}{0x4800\_0000} \approx 2nV$$

If the FIR is enabled, the calculation can involve a few additional factors. The first step is to calculate how many bits the FIR will add to the datapath. This depends on the number of taps (N_{taps}) and the coefficients (coeffi); another right shift, PP_SHIFTR, can be used to ensure the data fits (see **"FIR Filter"** on page 301 and **"Post processor right shift"** on page 305 for more details).

$$FIR_{ADJ} = \log 2\left(\sum_{i=0}^{N_{taps}} coeff_i\right)$$

For many useful filters, FIR_{ADJ} will be 0.

To ensure the Result fits the datapath, set PP_SHIFTR (S_{PP})

$$S_{PP} = [FIR_{ADJ}]$$

The complete equation for Co_{max}, including the effect of the filter:

$$Co_{max} = M \times U \times 2^{S_L + N_1 \log 2(DR_1) - S_R + N_2 \log 2(DR_2) - S_{RR} + FIR_{ADJ} - S_{P}}$$

Note: The automatic gain control allows one to change G_{PGA}, G_{cap}, and S_L. As long as increases in GPGA between GAINLVL_STRUCTS are offset by decreases in S_L, the same constants can be used when doing voltage conversions.

Caution: V_{range} is merely a constant used for calculation. The actual input voltage is restricted such that the output of the PGA does not exceed 1.2 V.

22.2.3.10 Offset correction values alignment

The Offset Correction (OCOR) values stored in SFlash are determined using specific Digital Channel (DCHAN) configurations. The OCOR is applied at the output of the decimator right shift (DEC_SHIFTR). Any changes to the DCHAN that affect the scale of this output require a corresponding change to either the DCHAN settings or the OCOR value in order to align them back to the same scale.

The OCOR is applied after DEC_SHIFTR, therefore consider the channel configuration up to DEC_SHIFTR (see **Figure 22-18**).

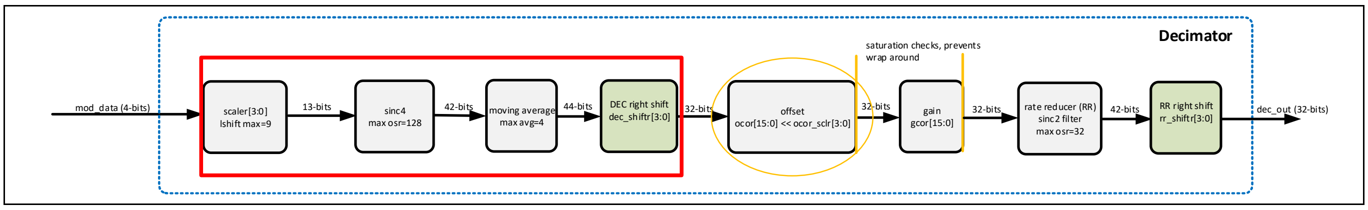

Decimator
mod_data (4-bits)
scaler[3:0] lshift max=9
13-bits
sinc4 max osr=128
42-bits
moving average max avg=4
44-bits
DEC right shift dec_shiftr[3:0]
32-bits
offset ocor[15:0] << ocor_sclr[3:0]
saturation checks, prevents wrap around
32-bits
gain gcor[15:0]
32-bits
rate reducer (RR) sinc2 filter max osr=32
42-bits
RR right shift rr_shiftr[3:0]
dec_out (32-bits)

Figure 22-18. Decimator right shifting stages

The OCOR values in SFlash are derived from the following specific channel configurations, resulting in specific resolutions at DEC_SHIFTR. All configurations use the following values:

- M = modulator data, ranges = 9 (–4 to 4)
- N = Number of sinc stages = 3 (sinc3)
- DR = Decimation ratio = 128

Table 22-7. Channel configuration for OCOR value in SFLASH for $DR_1 = 128$

Gain			SHIFTL (S_L)	DEC_SHIFTR (S_R)	Resolution of OCOR[a)] (nV)
Profile	Analog Gain (G_{ana})	Moving Average (U)			
0.5	0.5	1	0	0	286.1
1	1	1	0	0	143.1
2	0.5	4	8	3	2.235
4	1	4	7	3	2.235
8	2	4	6	3	2.235
16	4	4	5	3	2.235
32	8	4	4	3	2.235
64	16	4	3	3	2.235
128	32	4	2	3	2.235
256	64	4	1	3	2.235
512	128	4	0	3	2.235

a) Assumes OCOR_SCLR = 0

Any changes to the channel configuration need to match the Resolution after DEC_SHIFTR so that the OCOR values in SFlash are on the same scale. The Resolution after DEC_SHIFTR can be calculated with the equation.

$$V_{LSB} = \frac{\frac{1.2V}{G_{ana}}}{\max(M) \times U \times 2^{S_L + N_1 \log 2(DR_1) - S_R}}$$

If the resulting resolution is different from the value in **Table 22-7**, SHIFTL and DEC_SHIFTR must be adjusted.

Note: When AGC is used, SHIFTL is applied at each gain level, but DEC_SHIFTR can only be applied at the DCHAN level. This means that an increase of SHIFTL at one gain level requires an increase of SHIFTL in all other gain levels.

- Example: DR = 64 instead of 128

The following calculation uses the gain profile of 2x.

$$V_{LSB} = \frac{\frac{1.2V}{0.5}}{4 \times 4 \times 2^{8 + 3\log 2(64) - 3}} = 17.9nV$$

The conversion is not aligned. Lowering DR_1 requires a correction.

$$\frac{17.9}{2.235} = 8$$

To recoup the lost factor of 8, use a smaller S_R value, or a larger S_L value. Because the S_L value is already somewhat large, choose to modify S_R.

$$V_{LSB} = \frac{\frac{1.2V}{0.5}}{4 \times 4 \times 2^{8 + 3\log 2(64) - 0}} = 2.235nV$$

If the factor had been 1/8 instead, use a smaller S_L value.

Applying this to the previous table gives **Table 22-8**.

Table 22-8. Channel configuration for DR_1 = 64

Gain			SHIFTL (S_L)	DEC_SHIFTR (S_R)	Resolution of OCOR[a] (nV)
Profile	Analog Gain (G_{ana})	Moving Average (U)			
0.5	0.5	1	3	0	286.1
1	1	1	3	0	143.1
2	0.5	4	8	0	2.235
4	1	4	7	0	2.235
8	2	4	6	0	2.235
16	4	4	5	0	2.235
32	8	4	4	0	2.235
64	16	4	3	0	2.235
128	32	4	2	0	2.235
256	64	4	1	0	2.235
512	128	4	0	0	2.235

a) Assumes OCOR_SCLR = 0

22.2.3.11 FIR Filter

Two digital channels (dchan0 and 1) contain a Finite Impulse Response (FIR) Filter.

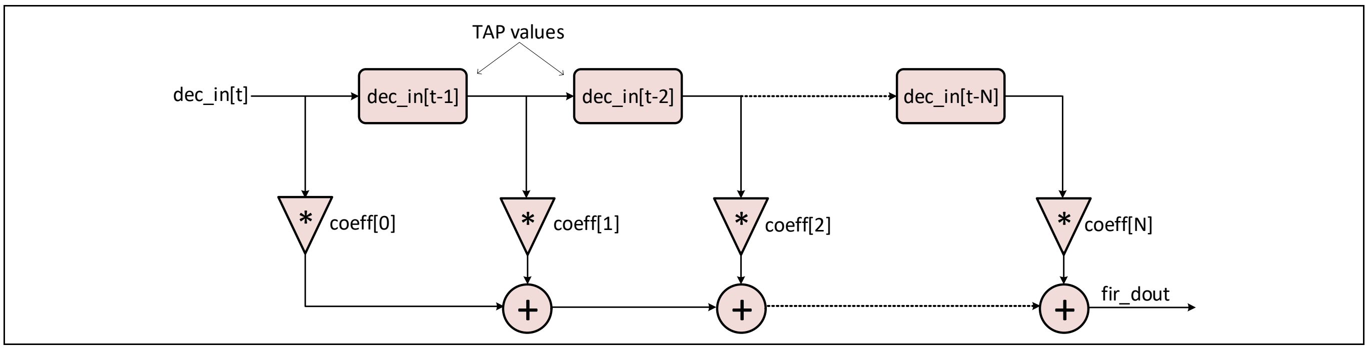


Figure 22-19. FIR Filter

From **Figure 22-19**, dec_in signifies the input data to the FIR from the decimator, while the [t] signifies is the instance in time. As data enters the FIR it is first multiplied by the initial coefficient value (coeff[0]); then it is added with the product (*) of the previous decimator data [t – 1] and the next stored coefficient value (coeff[1]), and so on. This continues until it reaches the number of TAPs enabled (N), which is a programmable register in PACSS_DCHANx_DCHAN_CTL. The maximum value is 64. The coefficient value (16-bits) and saved tap values (32-bits) are stored in separate SRAMs.

Note: The FIR filter registers (FIR_STRUCTx) are actually an SRAM, not registers. This means that they cannot be accessed by SWD while the ADC is running.

The coefficient values can range from $-1+1/2^{15}$ to $1-1/2^{15}$ with a resolution of $1/2^{15}$. The coefficient uses twos-complement notation and the decimal point is fixed to the right of the most significant bit.

Examples: –0.25 = 0xE000 and 0.25 = 0x2000

Figure 22-20 shows a state machine of the FIR filter.

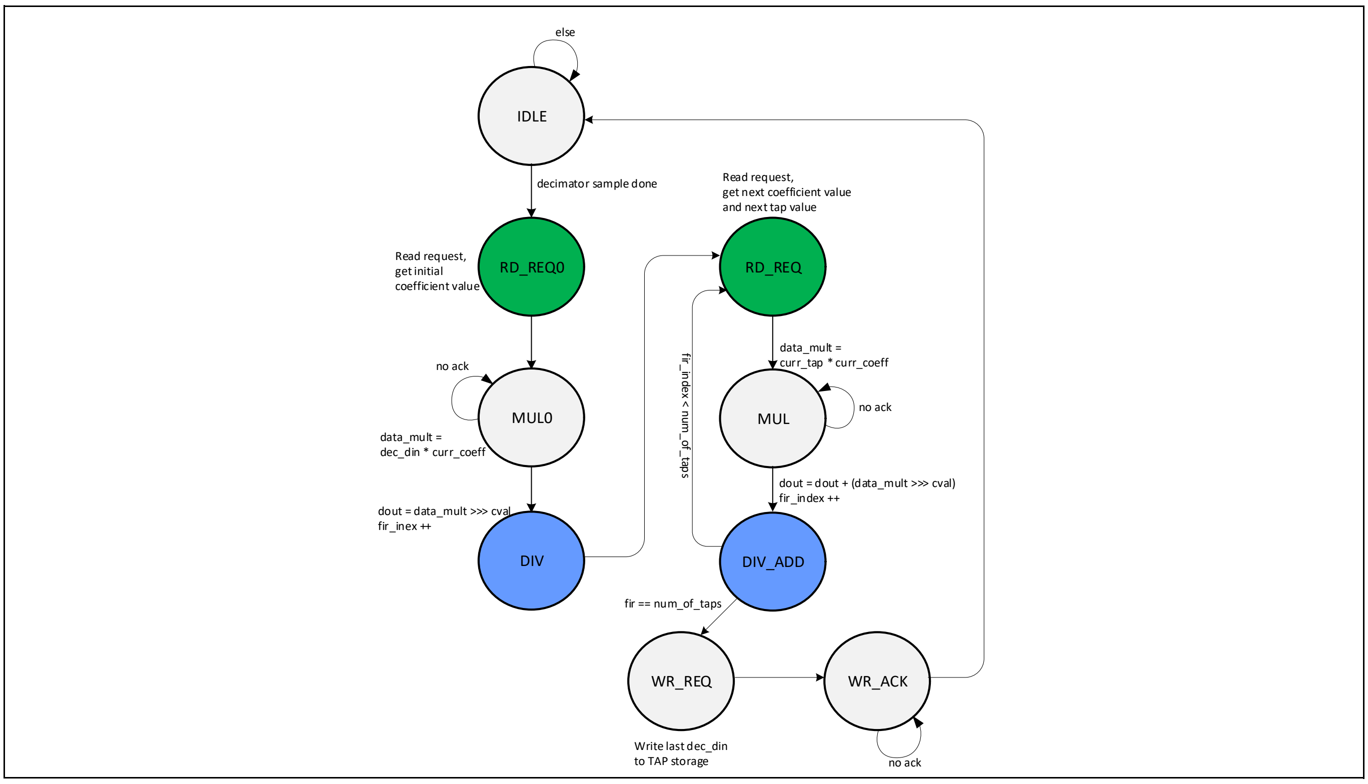

else
IDLE
decimator sample done
Read request, get initial coefficient value
RD_REQ0
Read request, get next coefficient value and next tap value
RD_REQ
no ack
MUL0
data_mult = dec_din * curr_coeff
data_mult = curr_tap * curr_coeff
MUL
no ack
fir_index < num_of_taps
dout = data_mult >>> cval
fir_inex ++
DIV
dout = dout + (data_mult >>> cval)
fir_index ++
DIV_ADD
fir == num_of_taps
WR_REQ
WR_ACK
Write last dec_din to TAP storage
no ack

Figure 22-20. FIR State Machine

Latency for data available at FIR output from reset: The general formula to calculate delay time/latency to get the first valid result at FIR output is as follows.

latency = (5 + [(FIR_taps + 1) × DR2 + (decimator_order – 1)] × DR) / Fdsm_clk

Example: Latency at 8 Ksps data rate:

decimator order = 3 (for example, sinc3)

DR = 64

DR2 = 6

FIR taps = 16

Latency = (5 + [17 × 6 + 2] × 64)) / 3.072e6 = 2.168 ms (typ)

[worst-case is 1% more, therefore: 2.190 ms]

22.2.3.12 Digital channel filters

The PACSS digital channel has two decimators (See **"Decimator"** on page 293) and an FIR filter (See **"FIR Filter"** on page 301). Over Sampling Rate (DR and DR2) of decimators is programmable. FIR filter coefficients and the number of taps are also programmable. **Figure 22-21** shows a high-level block diagram of the digital channel.

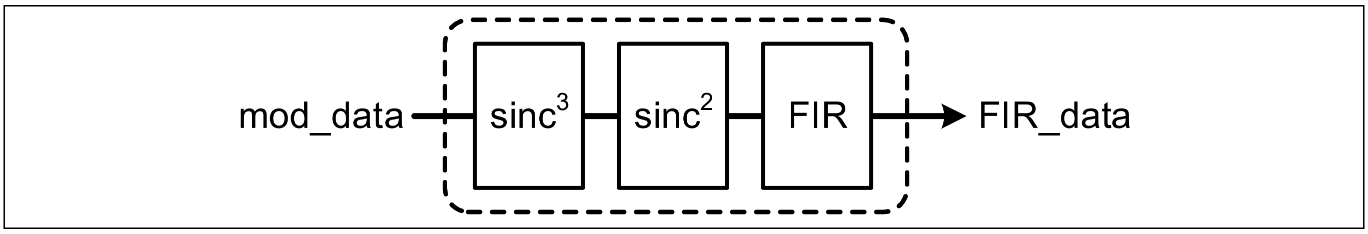


Figure 22-21. High-level block diagram of digital channel

Decimators, also known as CIC filters are used to filter quantization noise of the Delta-Sigma modulator block diagram of sinc3 and sinc2 decimators are shown in **Figure 22-22**.

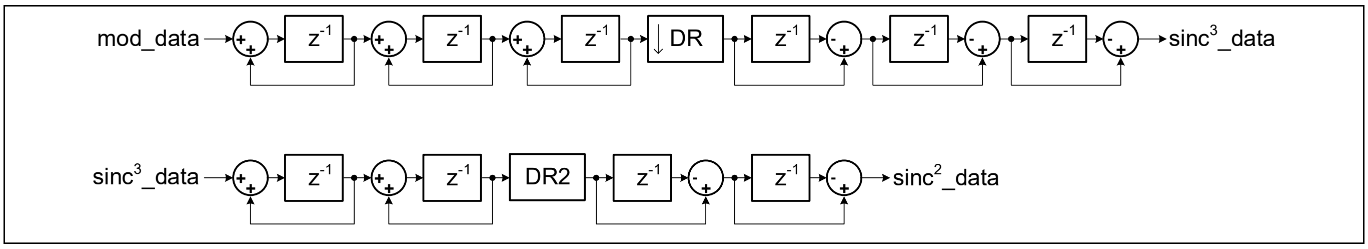


Figure 22-22. Block diagram of sinc3 and sinc2 decimators

The z-domain transfer function of an Lth-order CIC filter is typically presented as:

$$H(z) = \left[\frac{1 - z^{-DR}}{1 - z^{-1}}\right]^L$$

where,

L is the filter order

DR is the Over Sampling Rate

The sinc response is equal to zero at integer multiples of the data rate. Meaning, the magnitude response shows notches at these frequencies. Therefore:

sinc3 has notches at N*Fs/DR [notches at 24 kHz, 48 kHz, …]

sinc2 has notches at N*Fs/(DR*DR2) [notches at 1 kHz, 2 kHz, 3 kHz, …]

where,

N is an integer number

DR is the Over Sampling Rate of the sinc3 decimator

DR2 is the Over Sampling Rate of the sinc2 decimator

Fs is the ADC sample clock rate

An example response for DR = 128 and DR2 = 24 cases is shown in **Figure 22-23**. ADC sample clock rate is Fs = 3.072 MHz. Data-rate of this configuration is 1 ksps.

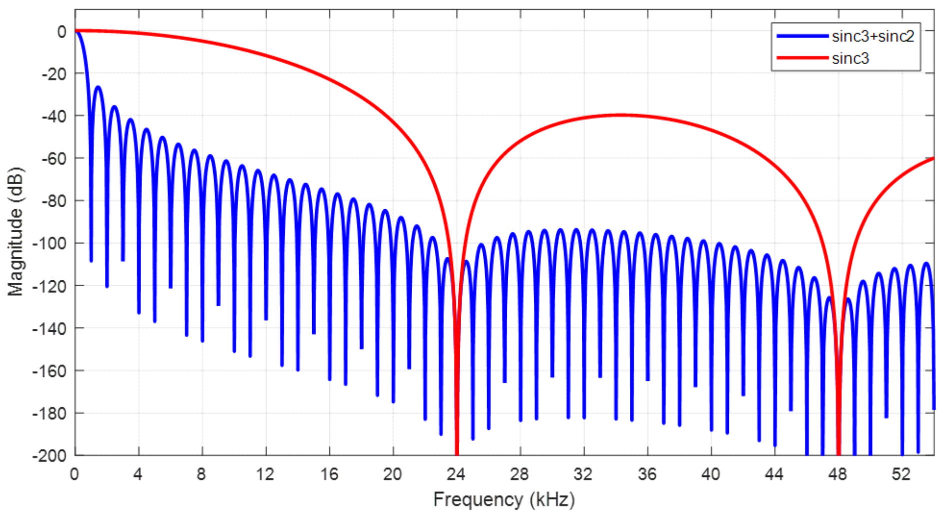


Figure 22-23. sinc3 and sinc3+sinc2 output comparison

In addition, an FIR filter can be used to further reduce bandwidth. FIR filter is fully programmable and has 64 taps. **Figure 22-24** is a general structure of the FIR filter block diagram.

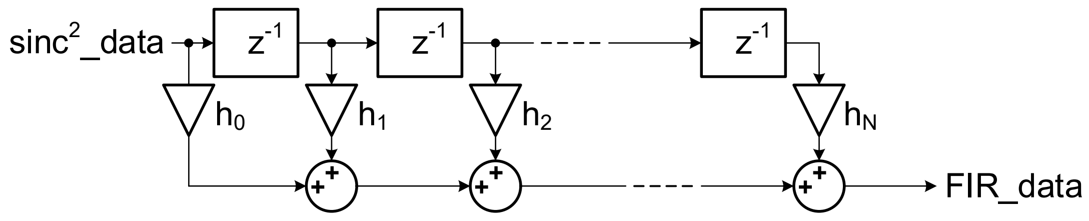


Figure 22-24. General structure of the FIR filter

Figure 22-25 shows the combined output with a 16-tap FIR filter. DR = 128 and DR2 = 24, ADC sample clock rate is Fs = 3.072 MHz. The coefficients used for the FIR filter are:

- [–0.00414 –0.01741 –0.02674 –0.0151 0.02896 0.1049 0.18756 0.24197
- 0.24197 0.18756 0.1049 0.02896 –0.0151 –0.02674 –0.01741 –0.00414]

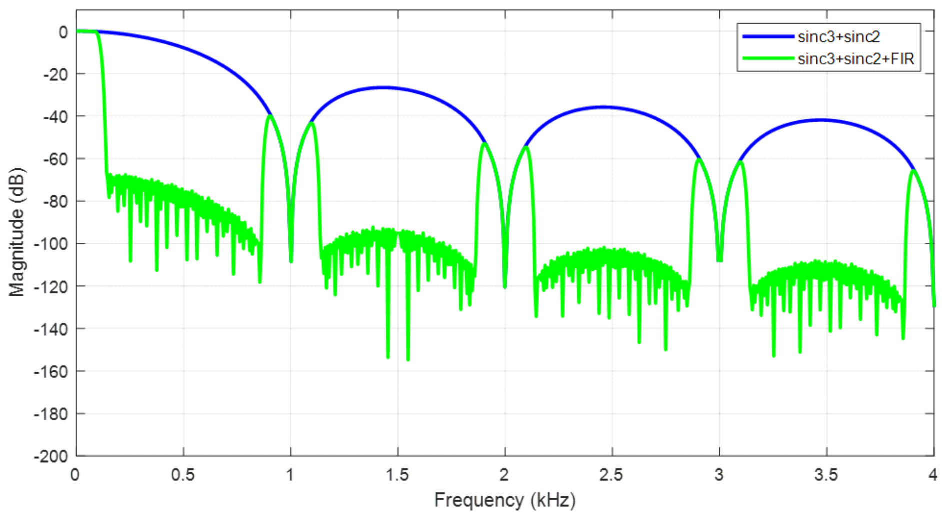


Figure 22-25. Combined output of the digital channel filters

22.2.3.13 Post Processor (PP)

Each digital channel contains its own post processor with a unique set of configuration registers (see **Figure 22-26**).

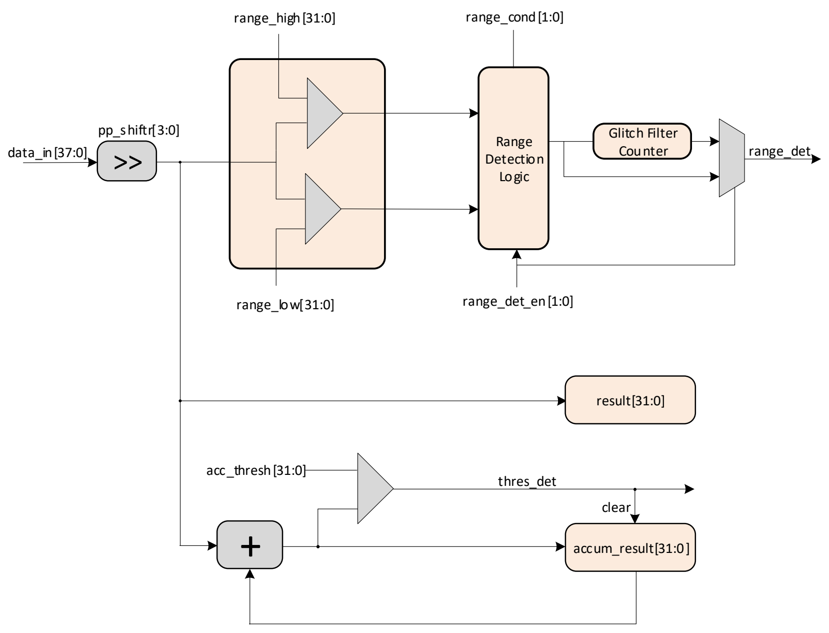


Figure 22-26. Post Processor

22.2.3.14 Post processor right shift

The 4-bit shift right value is used to make binary point alignment adjustments and to avoid truncation of the most significant bits from the FIR output. The right shift amount is determined by the number of FIR taps and coefficient values.

The first step is to calculate how many bits the FIR will add to the data path. This depends on the number of taps and the coefficient values, and is used to prevent the stored result from exceeding 32 bits.

FIR_DP_ADJ = log2(coeff[0] + coeff[1] + ... + coeff[n])

If the bit width of the data path plus FIR_DP_ADJ is greater than 32, PP_SHIFTR must be set. The data path equation at this point is:

DP_BW = log2(qlev) + SHIFTL + N × log2(DR) + 2 × log2(DR2) – DEC_SHIFTR – RR_SHIFTR

Therefore:

PP_SHIFTR = DP_BW + FIR_DP_ADJ – 32

22.2.3.15 Range detection

Range detection allows result comparison with two programmable signed threshold values without CPU involvement. It is defined by two 32-bit threshold values and a mode field for selecting one of the four possible modes. The range detection modes are:

- Below Low threshold (result < Lo)
- Inside range (Lo <= result < Hi)
- Above high threshold (Hi <= result)
- Outside range (result < Lo || Hi <= result)

There is also a range counter mode that can be enabled to filter out glitches. In this mode, an 8-bit count value is also set with one of the range detection modes. The count increments for every sample that triggers range detection event and decrements for every sample that does not trigger a range detection event.

22.2.3.16 Data storage

Each digital channel consists of a result, result tag, and accumulated result register. The result register is updated when the digital channel completes an acquisition. This may be at the end of the decimator conversion, the end of FIR filtering, or the end of post processing depending on how the channel is configured.

The result tag register contains the 7-bit ECC parity. This is the same ECC parity calculation used for the SRAM except that the address is not used. The same parity calculation can be used to with the address set to zero.

The accumulated result register accumulates signed acquisitions until a configurable threshold is reached (PACSS_DCHANx_ACC_THRESH.ACC_THRESH). When the threshold is reached the register resets to zero. The accumulated results register can also be written, so that its value can be reset at any time. The accumulated result occurs before the moving average calculation. To obtain an accumulated result the post processor must be enabled.

22.2.3.17 Interrupts and output triggers

Each digital channel contains the following interrupts (PACSS_DCHANx_INTR register):

- Range (RANGE_INTR)
  - The range interrupt is set whenever the measured value is meets the criteria for one of its four modes. The same interrupt is used if the glitch filtering counter is enabled. The range interrupt does require the post processor to be enabled.

  a) If RANGE_DET_EN = 01: When condition is met an interrupt is set; otherwise, nothing happens.

  b) if RANGE_DET_EN = 10: When condition is met, range detection counter is incremented, if less than RANGE_CNT_VAL.

  c) When condition is not met counter is decremented, if less than RANGE_CNT_VAL. When counter reaches RANGE_CNT_VAL interrupt is set. Write with '1' to clear bit.
- Accumulated Threshold (ACC_THRESH_INTR)
  - The accumulated threshold interrupt is set whenever the accumulated result register reaches the configured threshold value, which also means the accumulated result register has been reset. This interrupt does require the post processor to be enabled.
- Saturate Interrupt (SATURATE_INTR)
  - The hardware sets this interrupt if a conversion result is prevented from overflowing in either the decimator or the post-processor when PP_CTL.SAT_EN = 1. Write with '1' to clear bit.
- Overload Interrupt (OVERLOAD_INTR)
  - The hardware sets this interrupt for each channel if the modulator output is all zeros or all ones. This is an indication that the modulator is overloaded. Write with '1' to clear bit.
- Hardware Trigger Collision Interrupt (HWT_COLLISION_INTR)
  - The hardware sets this interrupt when the HW trigger signal is asserted while the DSM is BUSY. Raising this interrupt is delayed to when the scan caused by the HW trigger has been completed, i.e. not when the preceding scan with which this trigger collided is completed. When this interrupt is set it implies that the channels were sampled later than was intended (jitter). Write with '1' to clear bit.
- Firmware Trigger Collision Interrupt (FWT_COLLISION_INTR)
  - The hardware sets this interrupt when FW_TRIGGER is asserted while the DSM is BUSY. Raising this interrupt is delayed to when the scan caused by the FW_TRIGGER has been completed, i.e. not when the preceding scan with which this trigger collided is completed. When this interrupt is set it implies that the channels were sampled later than was intended (jitter). Write with '1' to clear bit.
- Overflow Interrupt (OVERFLOW_INTR)
  - The hardware sets this interrupt when it sets a new DATA_VAL_INTR while that bit was not yet cleared by the firmware. Write with '1' to clear bit.
- Data Valid (DATA_VAL_INTR)
  - After data is stored in the result register or the accumulated result register, the data valid interrupt will be set. If moving average is enabled the data valid interrupt will not be set until the moving average sample count is reached. Enabling the post processor is not required for this interrupt, but it is required to get an accumulated result. There is also a data valid hardware trigger that pulses once the data is valid and ready to be read. Write with '1' to clear bit.

Global configuration (pacss_mmio) contains the following interrupt (PACSS_MMIO_INTR register):

- AGC Gain Level Change Interrupt (AGC_GLVL_CHG)
  - The hardware sets this interrupt when a gain level change occurs. Write with '1' to clear bit.

22.2.4 Automatic gain control

The current channel has an automatic gain control (AGC), which allows a large dynamic range that enables measuring large starting currents or small battery-off currents. The gain of the current channel ranges from 1 to 512. Automatic gain can be disabled and set to a fixed value. A scaler between the modulator and decimator adjusts the weight of modulator data based on gain so the LSB of the data going into the decimator is always 0.715 mA. The gain of other channels is static instead of dynamic. Static gain is typically set to '1' but any value from 0.5 to 512 in powers of 2 can be used. The HV input channels have resistor voltage dividers that attenuate the input voltage. The nominal divider ratio is 24x (28.8 V full-scale) with an optional value of 16x (19.2 V full scale).

The AGC can automatically control the gain of either channel. AGC circuits typically monitor either the input voltage or output of the PGA and increase gain when signal amplitude is below a certain threshold or reduce gain when above another threshold. This function can also be achieved with a low-resolution A/D converter with digital comparison for gain selection.

For battery monitoring and management applications, the AGC is configured to operate with the primary current channel as shown in **Figure 22-27**. This functions by measuring the analog channel input voltage with an ADC and increasing or decreasing gain when programmable thresholds are reached. The output of the gain control comparators is used by a look-up table to set scaler, PGA, and modulator gain settings. To minimize input referred noise, it is advisable to configure the gain table to increase PGA gain before increasing modulator gain.

The PACSS uses a novel ADC implemented with a fast decimator providing 8–10 bit resolution, which is compared against programmable thresholds to decide when to increase or decrease gain. Gain is typically decreased when amplitude approaches 75% (PACSS_MMIO_AGC_CTL0.HI_THRESH) of full-scale and increased when amplitude falls below 25% (PACSS_MMIO_AGC_CTL0.LO_THRESH). As gain is increased or decreased, a table of PGA and modulator gain parameters is indexed to set overall gain as desired.

The highlighted AGC system is shown in **Figure 22-27** controlling a simplified channel.

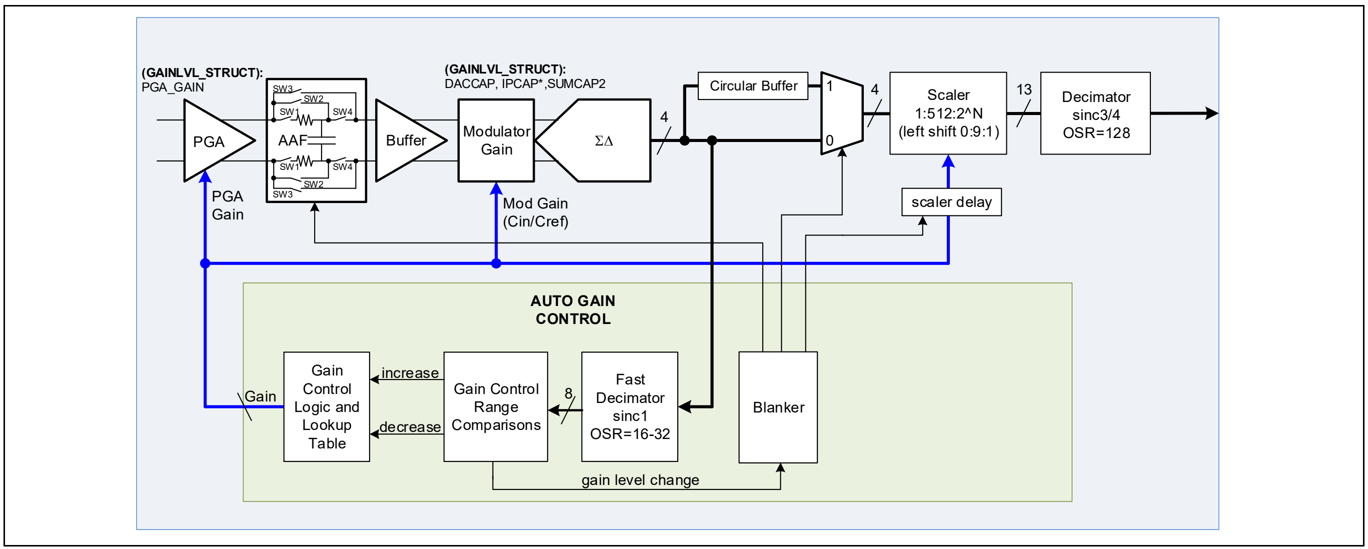


Figure 22-27. Automatic Gain Control (AGC) System

The AGC consist of 10 gain levels: 1 to 512. Each gain level contains register structure (GAINLVL_STRUCT) in the register map to provide programming of PGA gain, a left shift scaler, modulator cap value configuration to adjust its gain, and offset and gain correction values. The modulator cap values for each gain settings is given in **Table 22-9**.

Table 22-9. Modulator Cap Value Configuration

Gain	PGA	MOD	Digital[a]	Actual MOD gain[b]	Gain correction	IPCAP1	DACCAP	FCAP1	IPCAP2	FCAP2	IPCAP3	FCAP3	SUMCA P1	SUMCA P2	SUMCA P3	SUMCA PFB	SUMCA PIN
0.5[c]	0x00	0.5	MA-4	0.44	1.1429	0xD	0x1F	0x39	0x4	0xA	0x4	0xD	0x7	0x7	0x7	0x7	0x7
1	0x00	1	MA-4	0.88	1.1429	0x1B	0x1F	0x39	0x4	0xA	0x4	0xD	0x3	0x3	0x3	0x7	0x3
2	0x00	0.5	MS-4	0.44	1.1429	0xD	0x1F	0x39	0x4	0xA	0x4	0xD	0x7	0x7	0x7	0x7	0x7
4	0x00	1	MS-4	0.88	1.1429	0x1B	0x1F	0x39	0x4	0xA	0x4	0xD	0x3	0x3	0x3	0x7	0x3
8	0x00	2	MS-4	1.75	1.1429	0x1B	0xF	0x39	0x4	0xA	0x4	0xD	0x3	0x3	0x3	0x3	0x3
16	0x01	2	MS-4	1.75	1.1429	0x1B	0xF	0x39	0x4	0xA	0x4	0xD	0x3	0x3	0x3	0x3	0x3
32	0x02	2	MS-4	1.75	1.1429	0x1B	0xF	0x39	0x4	0xA	0x4	0xD	0x3	0x3	0x3	0x3	0x3
64	0x03	2	MS-4	1.75	1.1429	0x1B	0xF	0x39	0x4	0xA	0x4	0xD	0x3	0x3	0x3	0x3	0x3
128	0x04	2	MS-4	1.75	1.1429	0x1B	0xF	0x39	0x4	0xA	0x4	0xD	0x3	0x3	0x3	0x3	0x3
256	0x05	2	MS-4	1.75	1.1429	0x1B	0xF	0x39	0x4	0xA	0x4	0xD	0x3	0x3	0x3	0x3	0x3
512	0x05	4	MS-4	3.50	1.1429	0x1B	0x7	0x39	0x4	0xA	0x4	0xD	0x3	0x3	0x3	0x1	0x3

a) MA-4: Moving Average of 4, MS-4: Moving Sum of 4.
b) MOD: Modulator attenuates the input by ~15% gain and Gain Correction (GCOR) corrects this attenuation.
c) Gain = 0.5 setting is for static gain range of voltage channel (ACHAN1).

The left shift scaler is used to normalize results so the LSB always has the same weight, which is accomplished by multiplying the modulator output by the scaler value; the scaler value is the same as Log2 (512/gain).

The AGC is always sourced from dchan0. In most use cases this will be the digital channel configured for current measurement. The other configuration options are set in the PACSS_MMIO_AGC_CTLx registers. This is where the thresholds are configured and the decimation ratio of the fast decimator is set. It is not required to use all 10 gain levels; this can be reduced by setting the MIN_LVL and MAX_LVL fields in the PACSS_MMIO_AGC_CTL0 register. The initial gain level for where the AGC will begin can be specified with the INIT_LVL field.

The gain levels must be configured from lowest gain to highest gain. When the result of the fast decimator is compared with the thresholds it will increment the gain level if the result is lower than the low threshold, and decrement the gain level if the result is higher than the high threshold.

For example: gain level 0 can be set to a gain value of 1 and gain level 9 can be set to a gain value of 512.

When the AGC makes a gain level change and new gains are applied to the PGA and the modulator, the analog system may require settling time before the modulator data is valid or accurate. The AGC provides blanking features to compensate for this. The AGC contains an 8-bit counter that starts to increment after the gain level changes. The counter counts DSM clock cycles. PACSS_MMIO_AGC_CTL1 contains three configuration count values:

- AAF_BLANK_CNT is used to short the anti-aliasing filter. Shorting the AAF reduces the settling time.
- DEC_BLANK_CNT disconnects the decimator from the modulator data path and enables the circular buffer. During this period the decimator will use previously stored data from the modulator. This will allow the gain level changes to settle and keep the digital channel in its previous state until the modulator data is ready to be sampled.
- SCALER_BLANK_CNT is the number of clock cycles after the gain level change before the new scaler value from the new gain level is applied. The goal here is to only use the new scaler when the new modulator data is ready to be converted.

The blanking period is determined by the greatest count value of the three. The AGC will not perform any new gain level changes during the blanking period.

Each digital channel can be configured to use the AGC's gain updates (PACSS_DCHANx_DCHAN_CTL.AGC_GAIN_EN). In the primary use case the digital channels configured for the current and voltage measurements will set this bit and the channels configured for diagnostic or temperatures measurements will not.

Figure 22-28 is an example of the AGC with current going from 0% to 100% of full scale (3000 A). As described in the previous sections, the scaler is used to normalize results so the LSB always has the same weight (0.715 mA), which is accomplished by multiplying the modulator by the scaler value; the scaler value is the same as Log2(512/gain).

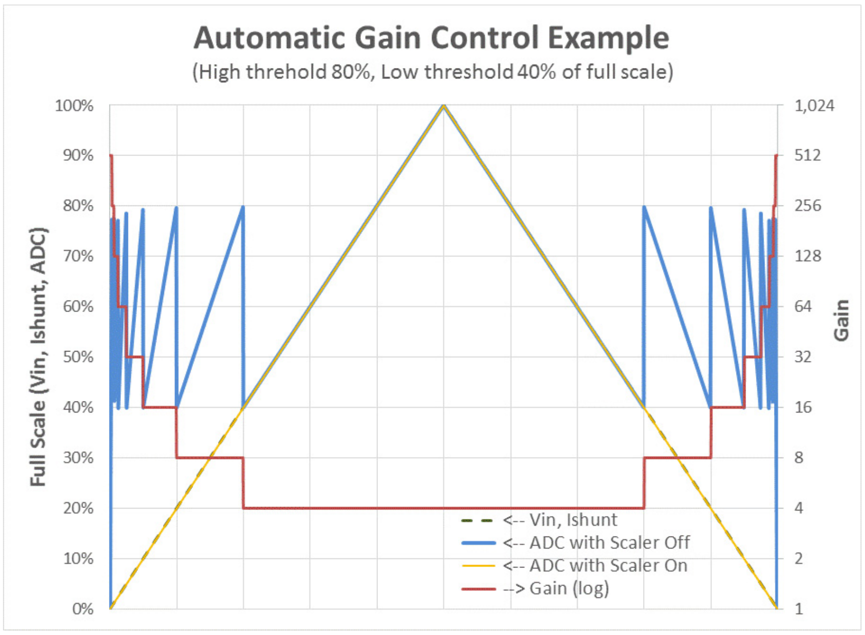


Figure 22-28. Automatic gain control example

Table 22-10 shows typical full scale current, resolution, and scalar values with various gain settings while the scaled LSB remains constant at 0.715 mA. As gain decreases, the modulator is shifted up (multiplied by the scalar).

Table 22-10. Typical Gain, Full Range Current Range, Resolution, and Scalar Values

Full Scale Range, Resolution, and Scalar vs. Gain (0.1 mΩ shunt an 1.2 V VREF)

Gain	Full Scale (±A)	Resolution (mA)	Scalar	SHIFTL Register Value
512	23.4375	0.715	1	0
256	46.875	1.431	2	1
128	93.75	2.861	4	2
64	187.5	5.722	8	3
32	375	11.444	16	4
16	750	22.888	32	5
8	1500	45.776	64	6
4	3000	91.553	128	7

Figure 22-29 illustrates scalar operation. A 4-bit modulator output from ±4 is scaled by 1 to 128. For example, when scaled by 128, ±4 becomes ±4 × 128 (= ±512).

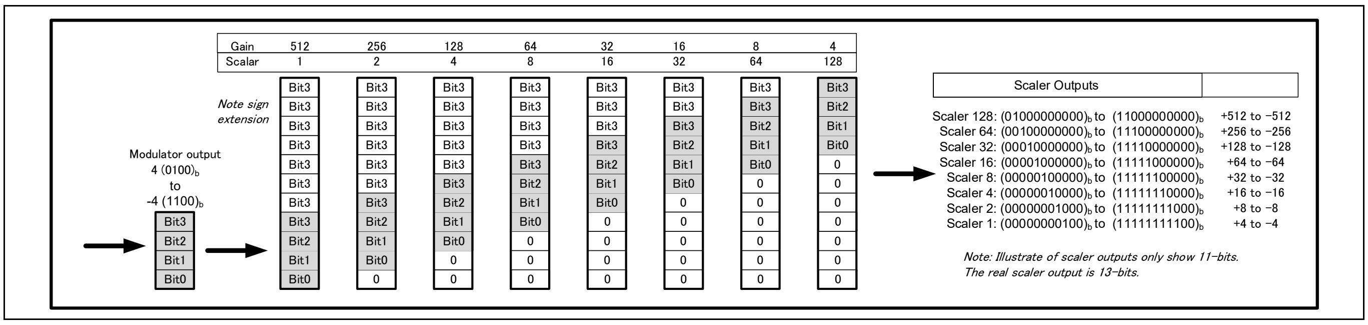

Gain 512 256 128 64 32 16 8 4
Scalar 1 2 4 8 16 32 64 128
Note sign extension
Modulator output
4 (0100)b
to
-4 (1100)b
Bit3
Bit2
Bit1
Bit0
Scaler Outputs
Scaler 128: (01000000000)b to (11000000000)b +512 to −512
Scaler 64: (00100000000)b to (11100000000)b +256 to −256
Scaler 32: (00010000000)b to (11110000000)b +128 to −128
Scaler 16: (00001000000)b to (11111000000)b +64 to −64
Scaler 8: (00000100000)b to (11111100000)b +32 to −32
Scaler 4: (00000010000)b to (11111110000)b +16 to −16
Scaler 2: (00000001000)b to (11111111000)b +8 to −8
Scaler 1: (00000000100)b to (11111111100)b +4 to −4
Note: Illustrate of scaler outputs only show 11-bits.
The real scaler output is 13-bits.

Figure 22-29. Scaler operation example

The AGC can be configured to increase or decrease gain by factors of two or more. Large steps can track large current changes faster by reducing the number of gain changes needed for tracking.

22.2.5 PACSS sequencer

The PACSS sequencer generates control signals for performing analog-to-digital conversion. Figure 22-30 shows the block diagram of the sequencer.

The PACSS has two sequencers that are associated with the two analog channels. When triggered, the sequencer loads the input pin selection of the first linked digital channel, then starts the decimator, which initializes the modulator to start the conversion. The modulator output data is transferred to the linked digital channel.

The two conversion modes supported are incremental (single sample) and continuous. When a conversion is finished, the sequencer is notified by the decimator.

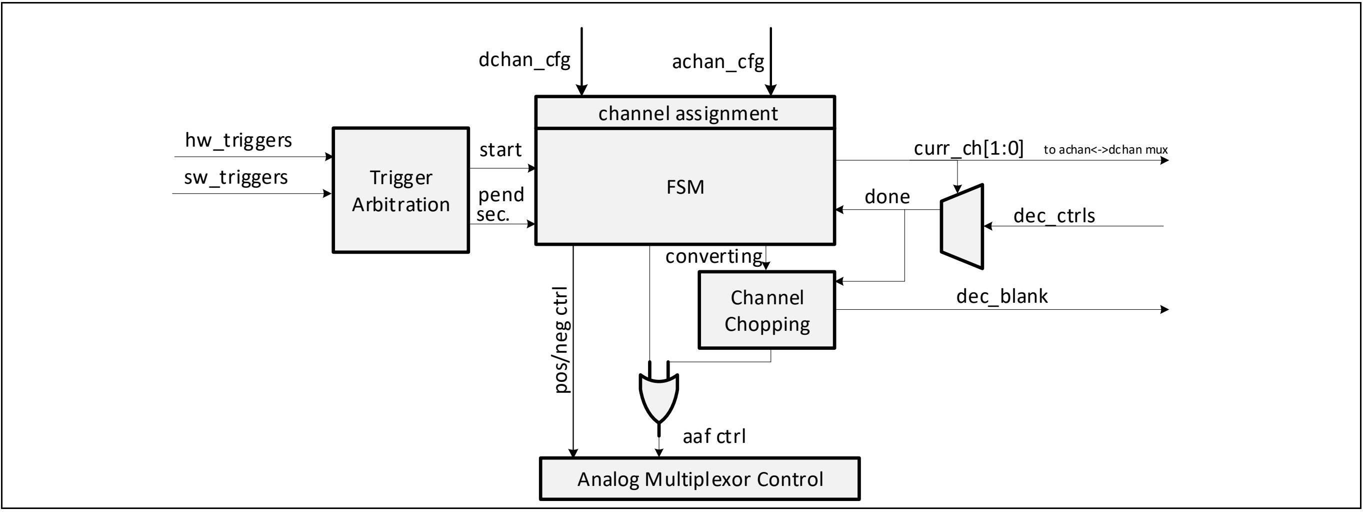


Figure 22-30. PACSS sequencer

A peripheral clock from the System-Wide Resources clock system is used for sequencer timing. The same clock is used for all ADC channels. The clock goes to the delta-sigma modulator (DSM) controller, which generates timing strobes with a finite state machine; these are used by other sections of the sequencer. The DSM controller also includes control and status registers, which are accessed from the µC and DMA channels using the AHB bus. The controller also generates interrupt and DMA requests.

ADC conversions are initiated from triggers generated by other sources. These sources include timers (TCPWM), input signals from GPIOs, software requests, and end-of-conversion commands from an active ADC. Triggers can start one or two ADCs – when triggering two ADCs, both ADCs start together. The two ADC channels have a channel controller, which generates timing for that channel. There are separate controls for the analog and digital portion of the channels. The analog timing signals control the delta-sigma modulator and chopping switches, while the digital timing signals clock data path registers including the decimator, finite impulse response filter, and comparators and accumulators. At the beginning of the conversion sequence, a two-cycle reset of the modulator clears state from previous conversions (the integrators in the modulator are initialized by this reset). The length of a conversion cycle requires the same number of clocks as the OSR setting. After the modulator is reset, the decimation filter needs to stabilize before output data is valid. The incremental measurement mode can be used to measure several different voltage sources for each conversion sequence while maintaining an aggregate sample rate. The sample rate can be adjusted by changing the OSR setting. If this mode is used to measure two different sources continuously, the sample rate for each signal is half of the single source rate.

PSoC™ 4 HV PA takes advantage of the incremental mode to create three channels with only two analog front-ends. The third channel can use either of the analog front-ends, which are not used between V/I measurements. This channel can be used for temperature measurement, to measure other signals, or for diagnostics. Diagnostics can include measuring power supplies, references, and even the same signal measured by the other channel. The main advantage of using shared analog front-ends is power reduction.

22.2.5.1 Triggering

The sequencers can be triggered via a hardware trigger (tr_start) or from a firmware trigger. The typical use case is for both sequencers to be triggered simultaneously but they can be triggered independently.

To set up a hardware trigger, one of the four trigger sources must be selected (PACSS_ACHANx_TR_CTL.PRIM_TR_SEL) and enabled (PACSS_ACHANx_TR_CTL.PRIM_TR_EN) via the PACSS_ACHANx_TR_CTL register. This register consists of primary and secondary fields. The primary trigger is used to start a conversion, while the secondary trigger is used to pend digital channels that are configured as secondary. The conversions of secondary channels always follow the conversion of primary channels.

The hardware triggers are rising edge detection, and have the option to be synchronized.

A firmware trigger can also be used to start a conversion via the PACSS_MMIO_START register. Using this register will start both sequencers simultaneous. To run the sequencers individually use the associated PACSS_ACHANx_START register.

If the primary triggered channel is in incremental mode, the secondary trigger must occur before the primary trigger for the secondary channel conversion to be appended to the primary conversion. If the secondary trigger occurs during the primary channel conversion, the secondary conversion is pushed until after the next primary conversion.

The trigger source for PACSS is selected in the PERI_TR_GROUP2_TR_OUT_CTLx[4:0] register.

Table 22-11 and **Table 22-12** provide the PSoC™ 4 HV PA PACSS trigger multiplexers and the multiplexer outputs

Table 22-11. PACSS trigger sources

PERI_TR_GROUP2_TR_OUT_CTL x[4:0]	Trigger source
0	Software trigger
1	TCPWM 0 overflow
2	TCPWM 1 overflow
3	TCPWM 2 overflow
4	TCPWM 3 overflow
5	TCPWM 0 underflow
6	TCPWM 1 underflow
7	TCPWM 2 underflow
8	TCPWM 3 underflow
9	TCPWM 0 compare match
10	TCPWM 1 compare match
11	TCPWM 2 compare match
12	TCPWM 3 compare match
13	PACSS data valid channel 0
14	PACSS data valid channel 1
15	PACSS data valid channel 2
16	PACSS data valid channel 3
17	GPIO input trigger 0
18	GPIO input trigger 1
19	GPIO input trigger 2
20	GPIO input trigger 3

Table 22-12. Trigger multiplexer outputs

Output	Trigger source
0	PACSS start conversion, digital channel #0
1	PACSS start conversion, digital channel #1
2	PACSS start conversion, digital channel #2
3	PACSS start conversion, digital channel #3

22.2.5.2 Channel selection

Each sequencer is assigned digital channels based on the PACSS_DCHANx_SMP_CTL.ACHAN_SEL setting. If more than one digital channel is assigned to a sequencer, the sequencer will convert the next assigned channel immediately following the conversion of its previous channel. By default, digital channels that are enabled are primary, setting PACSS_DCHANx_DCHAN_CTL.SEC_EN changes them to secondary.

For example, consider that dchan0 and dchan2 are assigned to sequencer 0 (seq0). When triggered, seq0 will first convert dchan0; on completion it will start the conversion on dchan2. If dchan2 is configured as a secondary channel via PACSS_DCHANx_DCHAN_CTL.SEC_EN, when seq0 is triggered (by a primary trigger) only dchan0 will convert.

When a secondary trigger occurs, the secondary channels assigned with the sequencer will go into a pending state. On the following primary trigger, the primary channels will convert followed by the secondary channels. In **Figure 22-31** all digital channels are configured as incremental. There should never be two primary channels assigned to the same sequencer when using the continuous conversion mode.

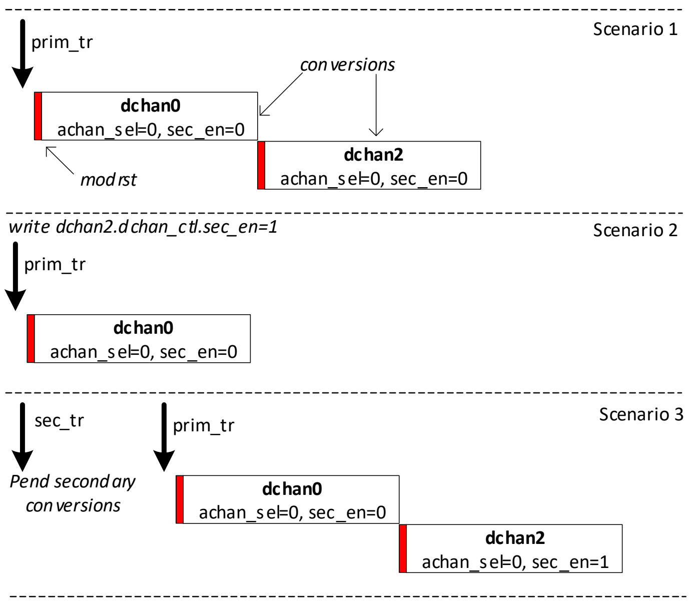


Figure 22-31. Primary vs. Secondary triggering

22.2.5.3 Input selection state machine

When the sequencer starts, it detects the first digital channel (lowest channel number) assigned to the sequencer. The next step is to load the channel configuration from the PACSS_DCHANx_SMP_CTL register. This includes the positive and negative pin selection, the channel selection delay, and the conversion mode of the digital channel. The channel selection delay provides additional delay between the pin selection and conversion if additional settling time is required for the analog system. Refer to the *PSoC™ 4 HV PA Registers TRM* for more information.

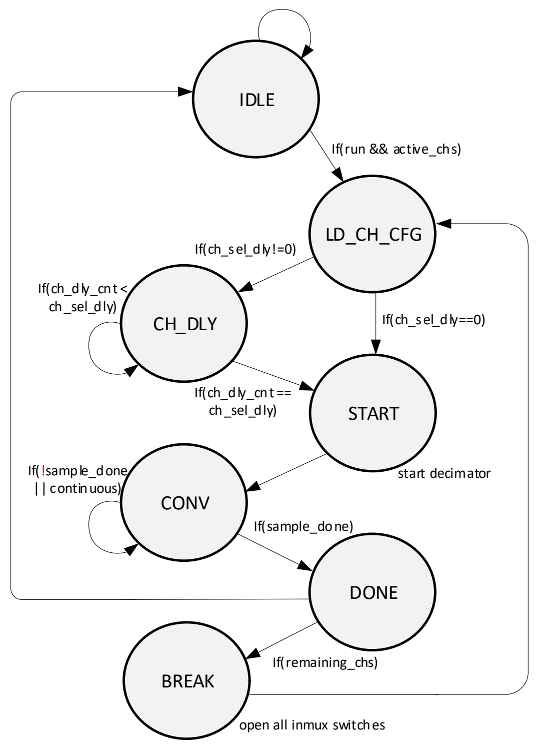


Figure 22-32. Sequencer FSM

The decimator is started in the next state, which resets the modulator and begins the conversion. If the conversion mode is set to incremental (PACSS_DCHANx_DEC_CTL.CONV_MODE = 0) after the sample is done, the decimator will assert sample done, and the sequencer will exit the conversion state. Otherwise, it will stay in the conversion mode state until the CONV_MODE bit is set to '0'. In the "done" state it will determine if other digital channels need to be converted or if the sequencer can go back to idle. The "break" state opens all the switches on the input mux before making another pin selection.

22.2.5.4 Sequencer timing

Figure 22-33 is a use case example that shows the sequencer timing when digital channel 0 (dch0) is assigned to sequencer 0 (seq0) and dch1 and dch2 are assigned to seq1. dch0 and dch1 are configured as primary channels with decimation ratios (DR) of 64 and rate reducer (DR2) of 6, for 8 ksps sample rate. dch2 is configured as a secondary channel and has a decimation ratios (DR) of 64 and rate reducer (DR2) of 1. All digital channel decimators are in the sinc3 mode.

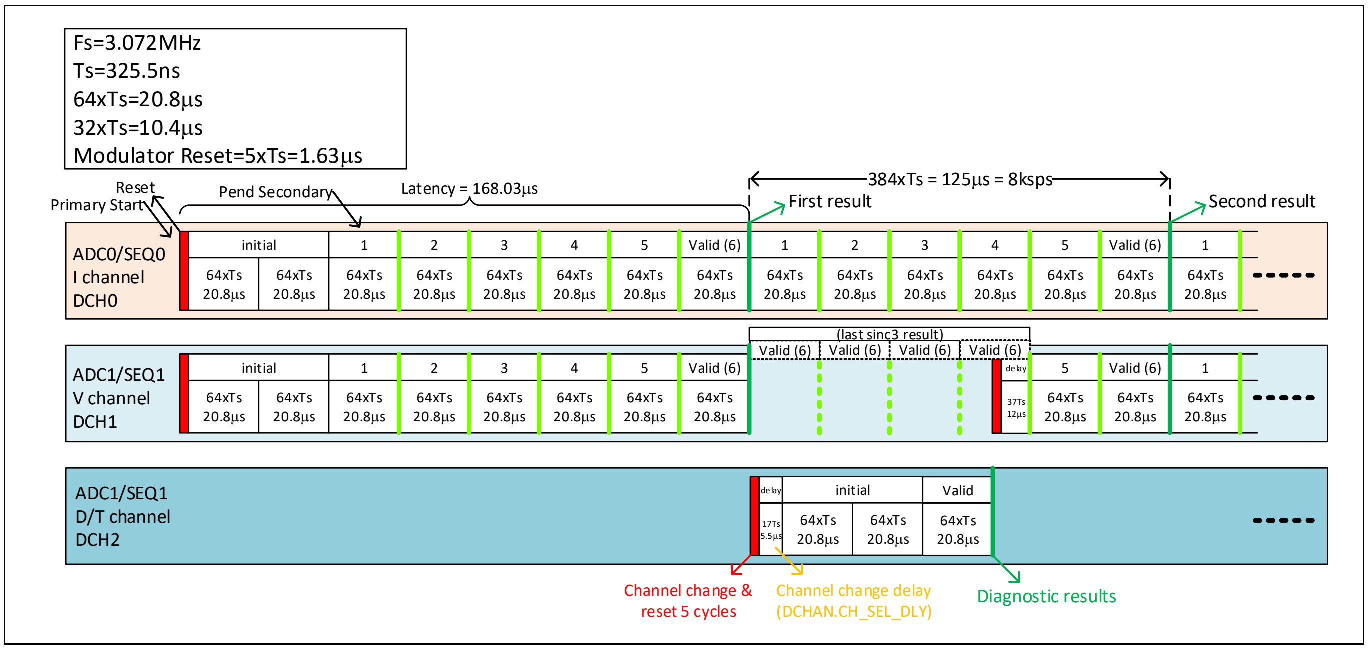


Figure 22-33. Sequencer timing

Both sequencers are configured to start from the same hardware trigger. After conversions begin on dch0 and dch1, a secondary trigger occurs. When seq1 is done with its active conversion (dch1), it loads the analog configuration of dch2 (input mux pin selection, and so on), then resets the modulator. The clock cycle delay, 17 is used here as an example and is inserted via the channel select delay it is a configurable delay used to add settling time between the mux selection and the conversion. The channel select delay should be set to the same value for any channels that need to be synchronized. There is an option to short the anti-aliasing filter during this delay to speed up settling time. After the channel select delay, the conversion begins (see **"Diagnostic Timing"** on page 316).

While the secondary channel is being converted on dch2, dch1's sinc3 decimator is frozen, and the last valid result of the sinc3 is reused until it resumes sampling from adc1. After the D/T (dch2) conversion is complete, the analog configuration of adc1 from dch1 is loaded, the modulator is reset, the channel select delay begins and the conversion resumes from the next sinc3 boundary, sampling actual data from adc1. During the D/T conversion the V channel result will be a merge of actual and frozen sinc3 data.

- Register settings example

PACSS_DCHANx_DEC_CTL.DR

PACSS_DCHANx_DEC_CTL.DR2

PACSS_DCHANx_SMP_CTL.CH_SEL_DLY

DCH0:	DR = 63, DR2 = 5, Sinc3, CH_SEL_DLY = 0
DCH1:	DR = 63, DR2 = 5, Sinc3, CH_SEL_DLY = 36
DCH2:	DR = 63, DR2 = 0, Sinc3, CH_SEL_DLY = 16

Note1: The value written to these registers will be plus 1 (for example, write 63 for 64).

Note2: Conversions with DR2 > 0 require 2 × DR + (2 × DR2 × DR) cycles to complete.

22.2.5.4.1 Conversion Mode Transition

1. The recommended behavior for stopping a Continuous channel from converting is to change the conversion mode from Continuous to Incremental, PACSS_DCHANx_DEC_CTL.CONV_MODE = 1 to 0.
2. The behavior of multiple DCHANs connected to the same ACHAN:
   a) Incremental mode: DCHANs will all convert in sequential order 0->3.
      Multiple Secondary DCHANs will convert in sequential order after receiving a Pend Secondary trigger. Conversions will be performed back to back.
   b) Continuous mode: Only the lowest-numbered DCHAN will convert
3. If the channel is running in Continuous mode and the mode is changed to Incremental, the last conversion will be in Incremental mode. This is important to note because if there are other DCHANs assigned as Primary channels to the same ACHAN they will convert after this last conversion, as described in 2.

22.2.5.4.2 Diagnostic Timing

1. The secondary trigger is applied at the end of Valid (6), which is the output of Rate Reducer (DR2 = 5).
2. Analog channel changes to programmed D/T input and modulator is reset. This takes five clock cycles.
3. A 17 clock cycle delay is inserted so that AAF, modulator, and other circuits settle (PACSS_DCHAN2_SMP_CTL.CH_SEL_DLY = 16).
4. During this time, AAF resistor should be shorted for five clock cycles for faster initial settling (PACSS_DCHAN2_SMP_CTL.AAF_SHORT_R_EN = 1).
5. D/T measurement is made with sinc3, DR = 63, DR2 = 0, incremental mode. This takes 3 × 64 clock cycles.
6. Analog channel changes to V input and the modulator is reset. This takes five clock cycles.
7. A 37 clock cycle delay is inserted so that AAF, modulator, and other circuits settle (PACSS_DCHAN1_SMP_CTL.CH_SEL_DLY = 36).
8. During this time, the AAF resistor should be shorted for five clock cycles for faster initial settling (PACSS_DCHAN1_SMP_CTL.AAF_SHORT_R_EN = 1).

In this timing scheme, in the V channel, the circular buffer data is fed to the decimator while the D/T measurement and channel switching is executed.

22.2.5.4.3 PACSS clock cycle equations

Table 22-13 shows the PACSS clock cycle equations and **Table 22-14** are example results of the clock cycle.

Table 22-13. PACSS clock cycle equations

Step	Cycle Count Equation	Comments
Input Trigger Processing	3	CLK_SYS (SYSCLK) constant
Sequencer Start	1	CLK_DSM constant
Sequencer Delay	1 + CSD	CSD = CH_SEL_DLY + 1, if CH_SEL_DLY = 0, CSD = 0
Reset & Start	4	Modulator and decimator resets time period
Decimator Initialization	(SINC_MODE – 1) × (DR + 1)	CLK_DSM equation
Decimator Conversion	if (DR2 > 0): 0, else: DR + 1	CLK_DSM equation
Moving Sum/Avg	(AVG_MODE > 0) × 2	CLK_DSM equation
Gain & Offset	(OCOR_EN × 3) + GCOR_EN × ((GVAL+2) + (!OCOR_EN))	CLK_DSM equation
Rate Reducer Start	3 × DR2_EN	CLK_DSM equation
Rate Reducer Initialization	((DR + 1) × (DR2 + 1)) × DR2_EN	CLK_DSM equation
Rate Reducer Conversion	((DR2 + 1) × (DR + 1)) × DR2_EN	DR2_EN = 1, IF DR2 > 0
FIR	((FIR_NUM_TAPS + 1) × 5 + 1) × FIR_EN	CLK_SYS Equation + 0.5 CLK_DSM (constant)
Post Processor	PP_EN × 6	CLK_SYS Equation
Output Trigger Processing	3	CLK_SYS constant

For example:

- Clock frequency
  - CLK_SYS = 49.152 MHz
  - CLK_DSM = 3.072 MHz
- Register value
  - CH_SEL_DLY = 16
  - SINC_MODE = 3 (sinc3 = 3, sinc4 = 4)
  - DR = 63
  - DR2 = 5 (DR2_EN = 1, if DR2 > 0)
  - AVG_MODE > 0 = 0 (disable)
  - OCOR_EN = 1 (enable)
  - GCOR_EN = 1 (enable)
  - GVAL = 15
  - FIR_EN = 1 (enable)
  - FIR_NUM_TAPS = 15
  - PP_EN = 1 (enable)

Table 22-14. Example of PACSS clock cycle

Step	Cycle Count Equation	Calculation	CLK_SYS	CLK_DSM
Input Trigger Processing	3	–	3	
Sequencer Start	1	–		1
Sequencer Delay	1 + CSD	1 + 16 + 1		18
Reset & Start	4	–		4
Decimator Initialization	(SINC_MODE – 1) × (DR + 1)	(3 – 1) × (63 + 1)		128
Decimator Conversion	if (DR2 > 0): 0, else: DR + 1	0		0
Moving Sum/Avg	(AVG_MODE > 0) × 2	0 × 2		0
Gain & Offset	((OCOR_EN × 3) + GCOR_EN) × ((GVAL + 2) + (!OCOR_EN))	((1 × 3) + 1) × ((15 + 2) + (0))		20
Rate Reducer Start	3 × DR2_EN	3 × 1		3
Rate Reducer Initialization	((DR + 1) × (DR2 + 1)) × DR2_EN	((63 + 1) × (5 + 1)) × 1		384
Rate Reducer Conversion	((DR2 + 1) × (DR + 1)) × DR2_EN	((5 + 1) × (63 + 1)) × 1		384
FIR	((FIR_NUM_TAPS + 1) × (5 + 1)) × FIR_EN	((15 + 1) × (5 + 1)) × 1	81	0.5
Post Processor	PP_EN × 6	1 × 6	6	
Output Trigger Processing	3	–	3	
Total clock cycle			93	942.5

- Example results

CLK_SYS time = 93 / (49.152 × 10^6) = 1.892 µs

CLK_DSM time = 942.5 / (3.072 × 10^6) = 306.803 µs

Total tine = 308.695 µs

22.2.5.4.4 DR, DR2 recommended for different sample rates

The sample rates are depended on DR an DR2 value. **Table 22-15** shows the recommended value for different sample rates.

Table 22-15. DR, DR2 recommended values (Register value. These values will be plus 1)

Sample rate (ksps)	DR	DR2
1	127	23
2	127	11
4	127	5
8	63	5
48	63	0

22.2.5.4.5 Registers list of persisting until next start of conversion

Certain registers only change after a start of conversion (SoC) is inserted. The list of these signals is as follows:

- PACSS_DCHANx_SMP_CTL register

  POS_PIN_SEL

  NEG_PIN_SEL

  BYPASS_AAF
- PACSS_DCHANx_SMP_REF_CTL register

  VREFH_SEL

  VREFL_SEL

  VREF_BUF_EN

  RS_PULLUP_SEL

  RS_PULLUP_EN

Because these registers do not change until the next SoC, the first result after SoC may get affected. This is a configuration transition that results in a discontinuity in the input conditions to the Channel that must settle out after conversion starts. The size of the glitch depends on previous and current input selection and the voltage level difference between these states (See **Figure 22-34**). If ADC is configured for a faster conversion (such as DR = 64, DR2 = 1), more than 1 sample may be affected. If AGC is enabled, the AGC gain level may change as a reaction to the glitch. AGC filter can be used to reduce this glitch.

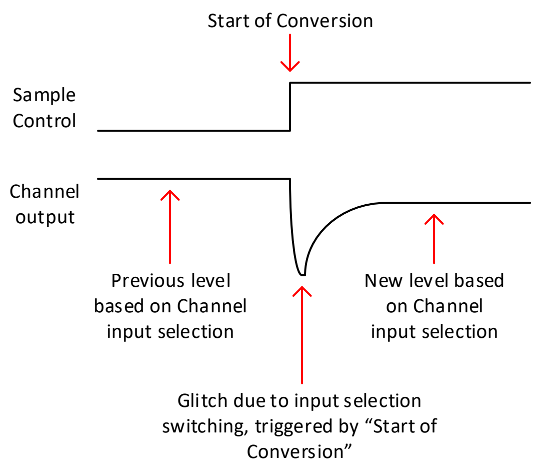

Start of Conversion
Sample Control
Channel output
Previous level based on Channel input selection
New level based on Channel input selection
Glitch due to input selection switching, triggered by "Start of Conversion"

Figure 22-34. Glitch by configuration transition

22.2.5.5 Startup timing requirements

The PACSS block has sequencing requirements at startup and wakeup as follows.

- Startup timing out of reset

1. Initialize and enable ADC sequencer and High Precision Band Gap Reference (HPBGR).
2. Wait 1000 µs.
   The sequencer must be enabled 1000 µs before enabling any analog channels to allow the reference currents to settle.
3. Initialize and enable analog channels (negative pump, LDO, PGA, Buffer, modulator, and so on).
4. Wait 3 µs.
5. Analog channel must be enabled for 3 µs before starting conversions to wait for AREF startup. Note that this startup time does not include AAF settling, which can take 64 µs (typ).
6. Initialize and enable digital channels.
7. Initialize and enable auto gain control (AGC) if necessary.
8. Start the conversion process.
9. Output delay is based on DR, DR2, and FIR filter configurations (See **Table 22-13**).

- Wakeup from deepsleep

1. Chip wakes up with AREF enabled.
2. Wait 25 µs for AREF to settle.
3. Enable HPBGR and analog channels.
4. Wait 150 µs for HPBGR to settle.
5. Start the conversion process.
6. Wait 60 µs for AAF settling (by using PACSS_DCHANx_SMP_CTL.CH_SEL_DLY = 180).
7. Output delay is based on DR, DR2, and FIR filter configurations (See **Table 22-13**).

22.2.5.6 Channel chopping

Channel chopping is where the sequencer swaps positive and negative inputs of the INMUX while also inverting the output of the modulator to cancel out any offset. When the inputs are not swapped, the output of the decimator is representation of Vin + Voffset. When the inputs (and modulator output) are swapped, the output of the decimator is Vin – Voffset. When both results are summed (at the following stages), the offset error is removed.

The sequencer generates a signal called "modbit"; when this signal is "1" the INMUX swaps its positive and negative input and the modulator data is inverted in the data path logic before entering the decimator. The PACSS_ACHANx_CHOP_CTL register configures channel chopping.

The channel chopping sample count register specifies the counted samples required before toggling the modbit. The sample count in this case, refers only to the decimation ratio of the sinc3/4 filter (not DR2). When the modbit toggles you can short the resistor on the AAF and/or activate the decimator's circular buffer "blank" for a set number of DSM clock cycles. Shorting the resistor on the AAF speeds up the settling time of the INMUX pin swap, and blanking the decimator reuses old modulator data while this settling completes.

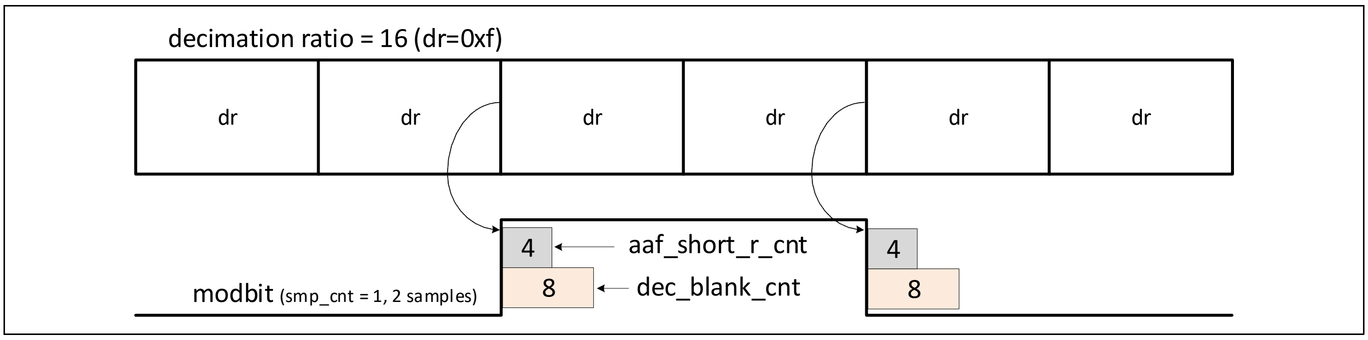

decimation ratio = 16 (dr=0xf)
dr
dr
dr
dr
dr
dr
4
aaf_short_r_cnt
8
dec_blank_cnt
4
8
modbit (smp_cnt = 1, 2 samples)

Figure 22-35. Sequencer Channel Chopping

The configuration shown in the **Figure 22-35** example is:

- PACSS_DCHANx_DEC_CTL.DR = 0xF
- PACSS_ACHANx_CHOP_CTL.SMP_CNT = 0x1, this counts 2 DR periods
- PACSS_ACHANx_CHOP_CTL.AAF_SHORT_R_CNT = 0x4
- PACSS_ACHANx_CHOP_CTL.DEC_BLANK_CNT = 0x8
- PACSS_ACHANx_CHOP_CTL.CH_CHOP_EN = 1

In **Figure 22-35** the modbit toggles after two DR periods; at this point the AAF resistor is shorted for four DSM clock cycles and the decimator is blanked for eight DSM clock cycles. This repeats after the next two DR periods.

Channel chopping is enabled with the PACSS_ACHANx_CHOP_CTL.CH_CHOP_EN bit, but it is only functional in the configurations where ACHAN0 is only assigned to one DCHAN. Otherwise, channel chopping will have unpredictable results.

22.2.5.7 Sequencer configuration rules

- An enabled analog channel should have at least one assigned primary digital channel.
- Secondary channels should always be set to incremental mode.
- If two primary channels are assigned to the same sequencer, their conversion modes should both be set to incremental.
- Primary channels should always have a lower digital channel index than secondary channels when assigned to the same sequencer. For example, an unsupported configuration is DCHAN0 set as the secondary while DCHAN1 is set as the primary.
- Channel chopping is only supported when ACHAN0 is only assigned to one DCHAN.

22.2.6 I/O components

22.2.6.1 Input multiplexer

The PSoC™ 4 HV PA PACSS input multiplexer (INMUX) selects between two differential input currents, differential input voltages including two high-voltage dividers, GPIO pads, on-die temperature sensors, an external NTC temperature sensor, and diagnostic voltages (see **Figure 22-36**). All analog signals can be supplied to either analog ADC, which improves diagnostics since both ADCs can measure and compare the same signals. Several integrated multiplexers facilitate diagnostics and channel switching without disturbing filters.

A sequencer selects the analog input signal by controlling the analog mux, connecting the input to an analog DSM system. Signals from GPIOs, on-chip power supplies and grounds, the high-voltage input voltage divider, and on-chip sensors and references can be selected. GPIOs not directly connected to the analog multiplexer can use an analog input bus called the analog multiplex bus (amuxbus). The amuxbus has two signals (amuxbusa and amuxbusb), which can connect GPIOs to ADC using software controlled analog switches inside each I/O. The analog DSM channel includes choppers and multiplexers in addition to the PGA, anti-alias filter, buffer amplifier, and DSM modulator (see **"Analog DSM system"** on page 279 for detailed information). This section also includes a temperature sensor.

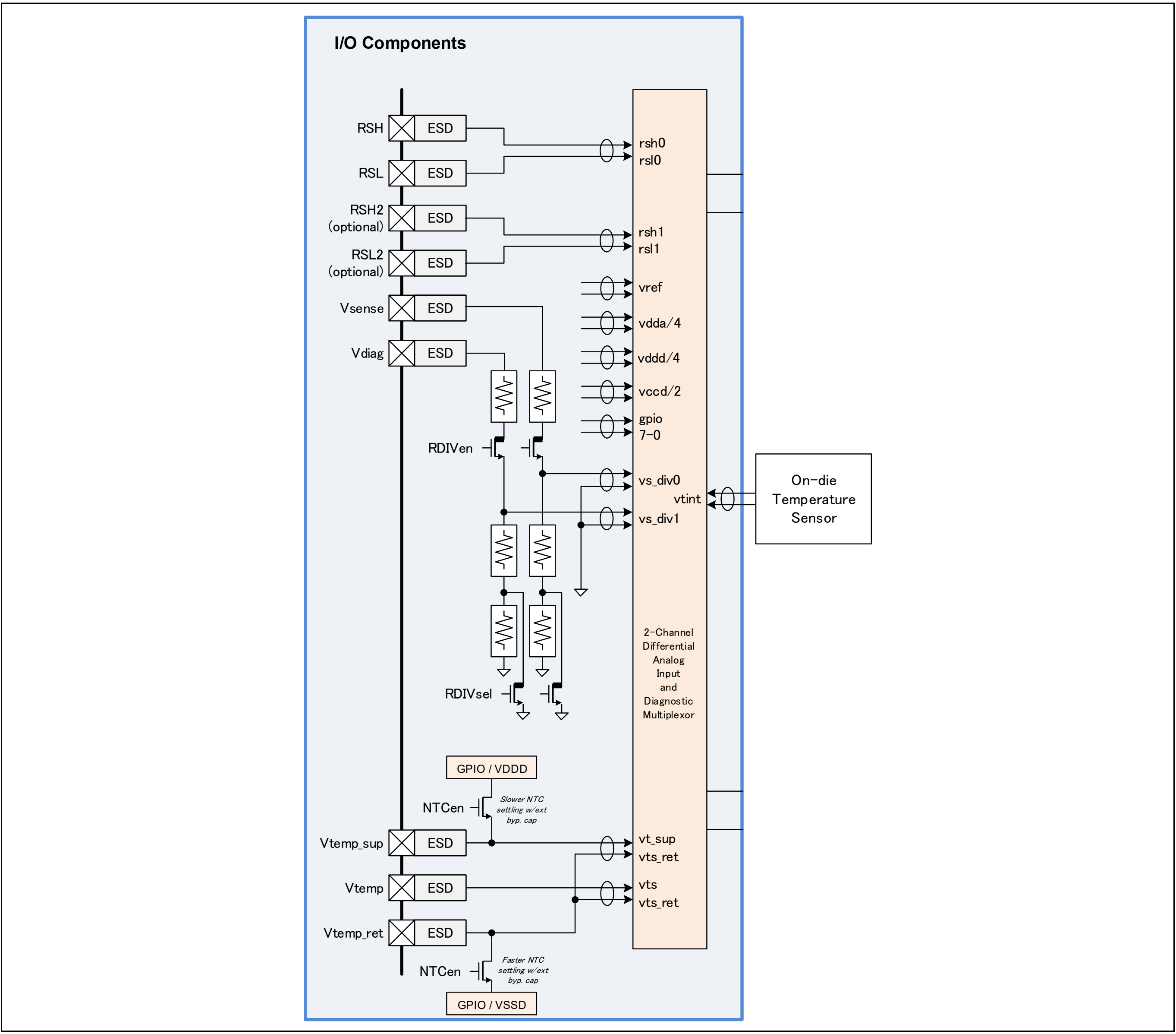


Figure 22-36. I/O Components

Figure 22-37 shows the INMUX diagram. The device supplies (vs*) go-through resistor dividers so they are scaled down to a level below the modulator reference. V_{DDA} and V_{DDD} are divided by 4, and V_{CCD} is divided by 2.

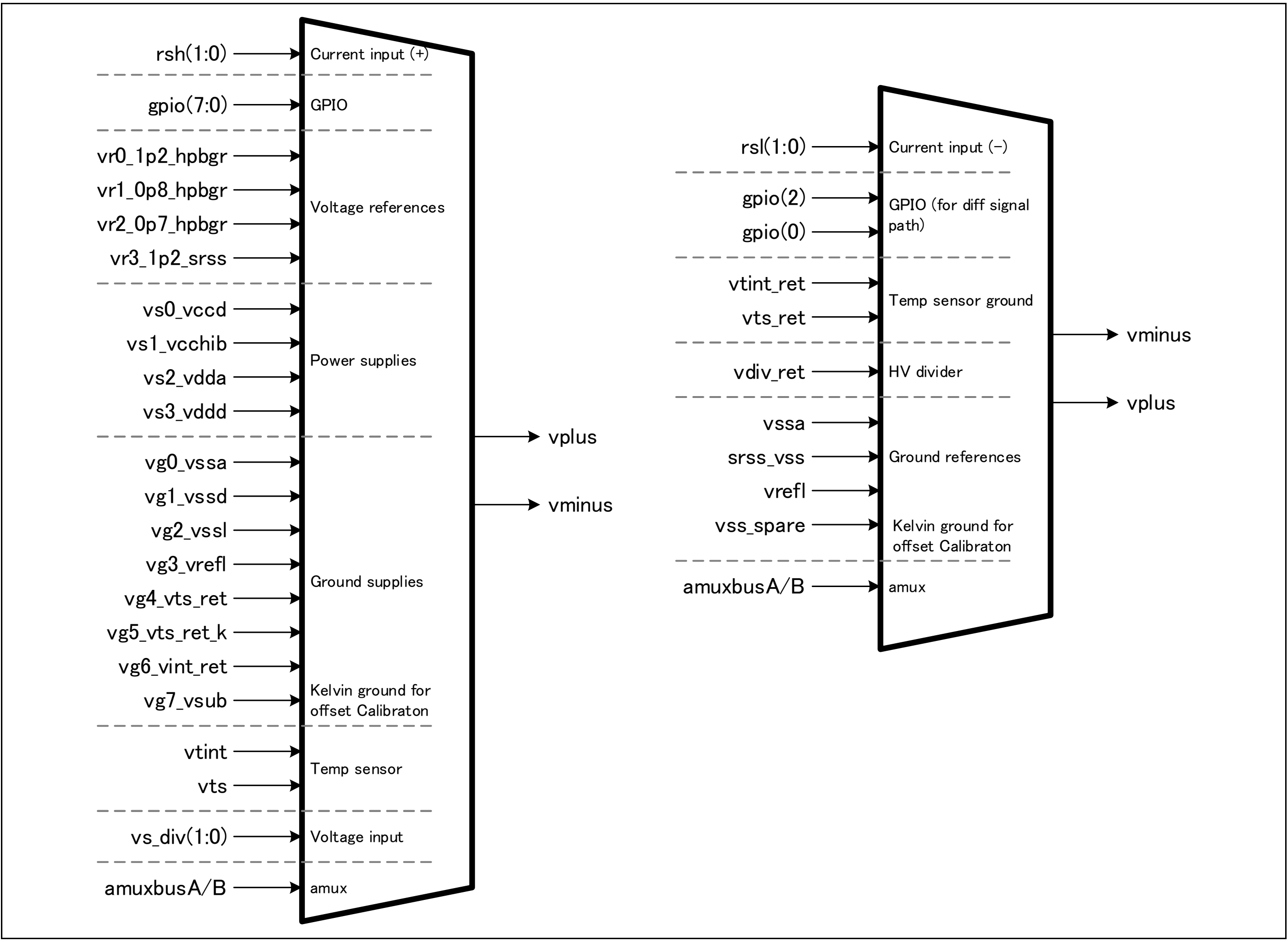


Figure 22-37. INMUX connectivity diagram

Table 22-16 and Table 22-17 show the INMUX truth tables.

Table 22-16. Positive pin selections (PACSS_DCHANx_SMP_CTL.POS_PIN_SEL)

POS_PIN_SEL <6:0>	Hex	Source	Description
00000XX	0x00	vs_div0	High-voltage Divider 0 (Vsense)
00001XX	0x04	vs_div1	High-voltage Divider 1 (Vdiag)
00010XX	0x08	vts	External (NTC) temperature
00011XX	0x0C	vtint	Internal temperature Sensor
0010000	0x10	vg0_vssa	Analog GND (VSSA)
0010001	0x11	vg1_vssd	Digital GND (VSSD)
0010010	0x12	vg2_vssl	LIN GND (VSSL)
0010011	0x13	vg3_vrefl	VREFL
0010100	0x14	vg4_vts_ret	External temperature sense GND
0010101	0x15	vg5_vts_ret_k	External temperature sense reference GND
0010110	0x16	vg6_vtint_ret	Internal temperature sense GND
0010111	0x17	vg7_vsub (vssa_k2)	Kelvin GND for offset Calibration
0011000	0x18	vs0_vccd	VCCD divided by 2
0011001	0x19	vs1_vcchib	VCCD divided by 2
0011010	0x1A	vs2_vdda	VDDA divided by 4
0011011	0x1B	vs3_vddd	VDDD divided by 4
0011100	0x1C	vr0_1p2_hpbgr	1.2 V VREF (HPBGR)
0011101	0x1D	vr1_0p8_hpbgr	0.8 V VREF (HPBGR)
0011110	0x1E	vr2_0p7_hpbgr	0.7 V VREF (HPBGR)
0011111	0x1F	vr3_1p2_srss	1.2 V VREF (SRSSHV)
01000XX	0x20	gpio0	P0.0
01001XX	0x24	gpio1	P0.1
01010XX	0x28	gpio2	P0.2
01011XX	0x2C	gpio3	P0.3
01100XX	0x30	gpio4	P0.4
01101XX	0x34	gpio5	P0.5
01110XX	0x38	gpio6	P0.8
01111XX	0x3C	gpio7	P0.7
100XXXX	0x40	rsh0	RSH
110XXXX	0x60	rsh1	RSH2
1111010	0x7A	amuxbusA	If selected, neg_pin_sel should be set to amuxbusB
1111011	0x7B	amuxbusB	If selected, neg_pin_sel should be set to amuxbusA

Table 22-17. Negative pin selections (PACSS_DCHANx_SMP_CTL.NEG_PIN_SEL)

NEG_PIN_SEL <4:0>	Hex	Source	Description
00XXX	0x00	vssa	Analog GND (VSSA)
01000	0x08	vss_spare	Reserved
01001	0x09	vrefl	VREFL
01010	0x0A	vss_srss	SRSS GND
01011	0x0B	vdiv_ret	HV divider GND
01100	0x0C	vts_ret	External temp sense GND
01101	0x0D	vint_ret	Internal temp sense GND
01110	0x0E	gpio0	P0.0
01111	0x0F	gpio2	P0.2
10000	0x10	rsl0	RSL
11000	0x18	rsl1	RSL2
11010	0x1A	amuxbusA	If selected, pos_pin_sel should be set to amuxbusB
11011	0x1B	amuxbusB	If selected, pos_pin_sel should be set to amuxbusA

Table 22-18 shows different input options and the corresponding positive and negative pin selections for these options.

Table 22-18. Pair of positive and negative pin selections

Positive Pin		Negative Pin	
Hex	**Source**	**Hex**	**Source**
0x00	vs_div0 (HV divider 0)	0x0B	vdiv_ret (Ground of HV divider)
0x04	vs_div1 (HV divider 1)	0x0B	vdiv_ret (Ground of HV divider)
0x08	vts (external temp sens)	0x0C	vts_ret (Ground of external temp sensor)
0x0C	vtint (internal temp sens)	0x0D	vint_ret (Ground of internal temperature sensor)
0x10	vg0_vssa	0x00	vssa
0x11	vg1_vssd	0x00	vssa
0x12	vg2_vssl	0x00	vssa
0x13	vg3_vrefl	0x00	vssa
0x14	vg4_vts_ret	0x00	vssa
0x15	vg5_vts_ret_k	0x00	vssa
0x16	vg6_vtint_ret	0x00	vssa
0x17	vg7_vsub (vssa_k2)	0x00	vssa
0x18	vs0_vccd	0x00	vssa
0x19	vs1_vcchib	0x00	vssa
0x1A	vs2_vdda	0x00	vssa
0x1B	vs3_vddd	0x00	vssa
0x1C	vr0_1p2_hpbgr	0x09	vrefl
0x1D	vr1_0p8_hpbgr	0x09	vrefl
0x1E	vr2_0p7_hpbgr	0x09	vrefl
0x1F	vr3_1p2_srss	0x0A	vss_srss
0x20	gpio0	0x00	vssa
0x24	gpio1	0x00/0x0E	vssa or gpio0 for differential signal
0x28	gpio2	0x00	vssa
0x2C	gpio3	0x00/0x0F	vssa or gpio2 for differential signal
0x30	gpio4	0x00	vssa
0x34	gpio5	0x00	vssa
0x38	gpio6	0x00	vssa
0x3C	gpio7	0x00	vssa
0x40	rsh0	0x10	rsl0
0x60	rsh1	0x18	rsl1

22.2.6.2 On-die temperature sensor

On-die temperature measurements are made with an internal temperature sensor by measuring bipolar V_{BE} at different current densities and calculating temperature. Two independent current references are used for diagnosis and redundancy. In addition to temperature, die stress also causes V_{BE} shifts but NPN and PNP bipolars respond differently to stress; therefore, arrays of both transistor types are included. **Figure 22-38** shows the on-die temperature sensor.

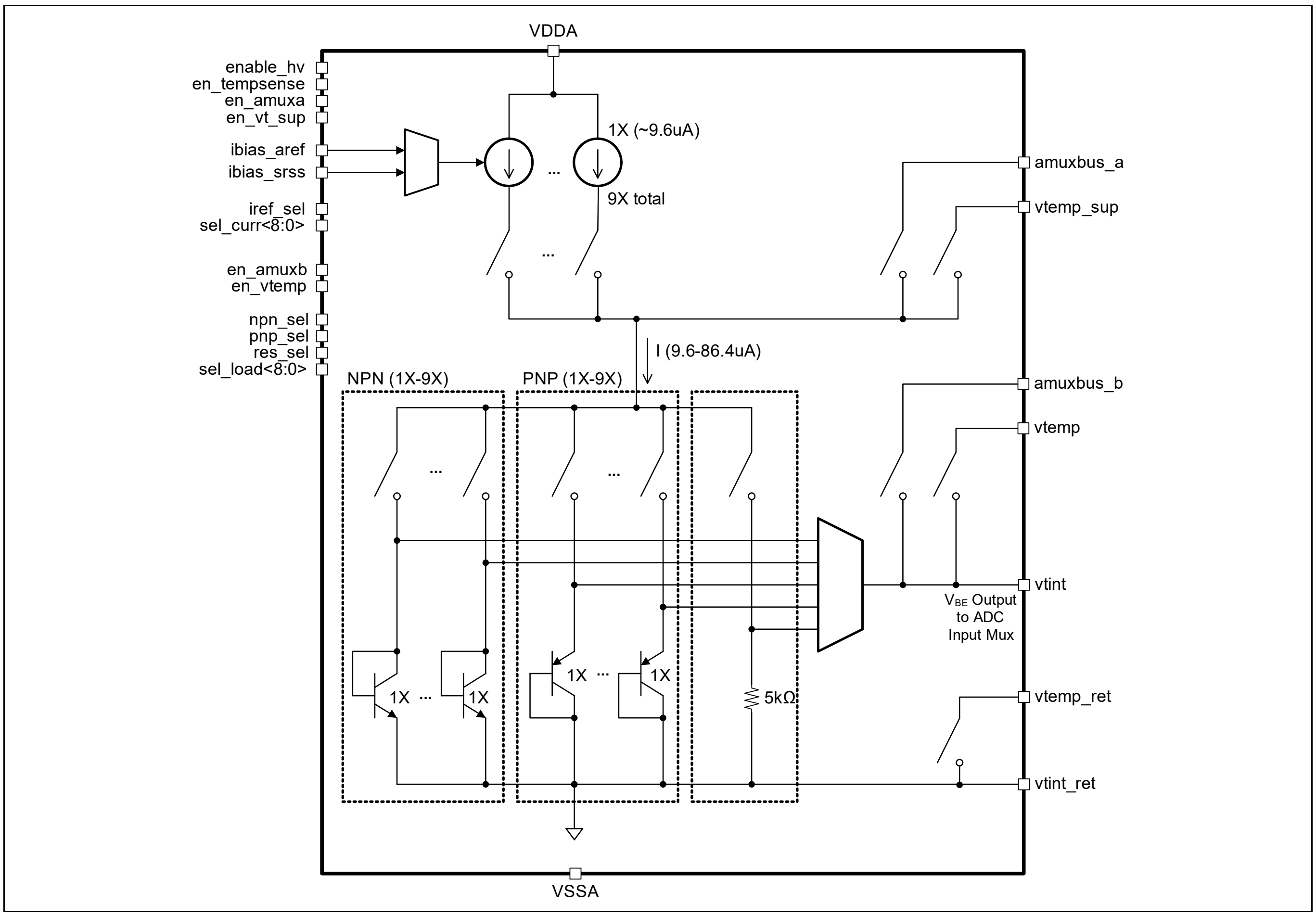


Figure 22-38. On-die temperature measurement

This sensor uses the principle that V_{BE} for a diode-connected bipolar transistor is proportional to the logn of junction current. Either one transistor with two different currents or two different size transistors with the same current value can be used. This sensor supports both schemes.

Temperature is directly proportional to the difference in V_{BE} voltage at two different current densities. The following equation shows how it works:

$$\text{Temperature } (^{0}K) \propto (V_{BE2} - V_{BE1}) \times 1 / \ln(I_{BE2}/I_{BE1}) \qquad (22.1)$$

V_{BE2} and V_{BE1} are base-emitter voltage at current densities of I_{BE2} and I_{BE1}

Note: Equation 22.6 shows the formula for calculating internal temperature.

This sensor supports sensing temperature with either one transistor measured with two different currents, or one current with two different size transistors. However, mismatch between bipolar devices are not calibrated. There are provisions to measure currents using a resistor to allow correction for current mismatch.

Suggested flow for temperature measurement:

1. Decide the flavor and number of bipolar devices to use.
2. Decide the low and high currents to be used.
3. Set the current to low, and using diagnostic channel, measure the voltage across the resistor, and record the value.
4. Set the current to high, and using diagnostic channel, measure the voltage across the resistor, and record the value.
5. Ratio of these two voltages will give the ratio of two currents.
6. Set the current to low, and using diagnostic channel, measure the V_{BE} voltage, and record the value.
7. Set the current to high, and using diagnostic channel, measure the V_{BE} voltage, and record the value.
8. Now that both V_{BE} and current ratio are known, the temperature can be calculated using the previously provided formula.

The sensor also supports external current and voltage monitoring. Selection of current source and V_{BE} monitoring is independent of each other. The amuxbuses can be used to supply current on amuxbus_a and monitor V_{BE} on amuxbus_b. Because the amuxbuses have many connections, excessive leakage at high temperature may affect their utility; therefore, a set of external connections directly to GPIOs is available. The set of GPIOs chosen is the three external temperature measurement pins – vtemp_sup, vtemp, and vtemp_ret.

The options for redundancy are as follows:

- Using different current source (SRSS vs AREF as current source)
- Using NPN or PNP type bipolar
- Using different ratios for generating delta V_{BE}
- Using different number of NPN or PNP units

22.2.6.3 External temperature sensor

The external temperature sensor uses a fixed resistor and a temperature-dependent resistor (generally a negative temperature coefficient thermistor) – the resistor and thermistor form a voltage divider whose output voltage is temperature dependent. When enabled, the External Temperature Sensor block provides power to the off-chip temperature-dependent divider (and a reference supply for the ADC) of $V_{DDD}/3$. **Figure 22-39** shows the block diagram of the external temperature sensor.

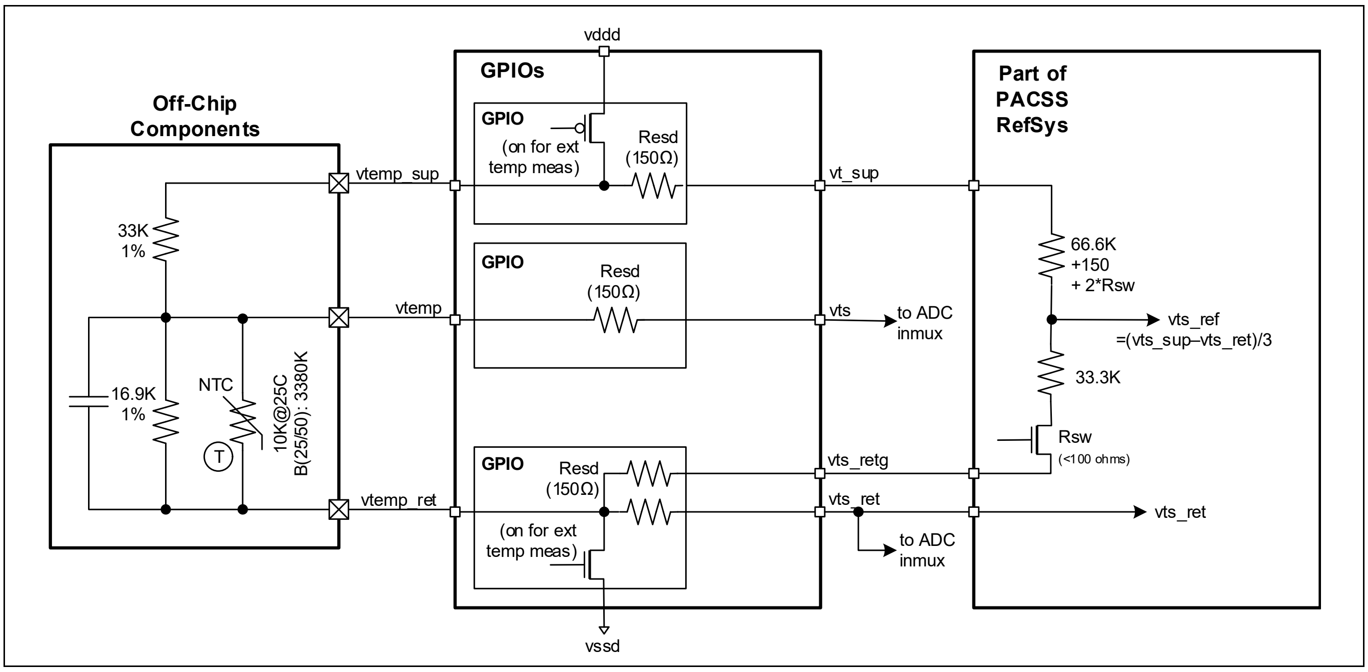


Figure 22-39. External temperature sensor

The off-chip voltage divider be selected to optimize temperature accuracy and will typically use a voltage divider comprised of a 33 kΩ and 16.9 kΩ fixed resistors (R) and a 10 kΩ nominal NTC thermistor RT (nominal NTC resistance is typically specified at 25°C). NTC resistance is found by supplying vts_ref as the reference input of the ADC, measuring vts voltage, which produces a ratio-metric value of vts with respect to vts_sup, and calculating RT.

Manufacturers provide NTC coefficients that can be used to convert resistance to temperature. Temperature can be calculated with either beta values (β) or Steinhart-Hart coefficients. Beta (β) values change as a function of temperature range and are only accurate to about ±5°C over a 200°C temperature range. They are calculated by measuring resistance and two temperatures, typically room temperature (RT25 is resistance at 25°C and T25 is 298.15°K) using the following expression with temperatures in kelvin:

$$\beta = \ln(RT25/RT) / ((1/T25) - (1/T)) \quad (22.2)$$

$$\ln(RT) = \ln(RT25) \times \beta \times ((1/T) - (1/T25)) \quad (22.3)$$

β can be used to convert resistance to temperature using the following equation:

$$T\ (°K) = 1 / (\ln(RT/RT25)/\beta + 1/T25) \quad (22.4)$$

The example circuit uses an NTC with a β(25/50) of 3380K (ln(RT25/RT50)/(1/(273 + 25) – 1/(273 + 50)) = 3380).

Steinhart coefficients are determined by measuring resistance at three temperatures and can be accurate to ±0.15% over a 200°C range. The equation for determining temperature from resistance is:

$$T\ (°K) = 1 / (A + B(\ln(R)) + C(\ln(R)^3) \quad (22.5)$$

22.2.6.4 HV input divider

The HV input divider is a voltage divider used on the VSENSE and VDIAG to scale battery voltage to levels compatible with the ADCs, so battery voltages can be measured. The HV input divider is part of high-voltage subsystem (see **"VDIVIDER (high-voltage divider)"** on page 357 for more information).

22.2.7 PACSS calibration

22.2.7.1 Channel OFFSET and GAIN calibration

The digital channels allow adjustments to offset and gain to correct cumulative errors in the analog circuits.The offset and gain is calibrated during test at multiple temperatures and for multiple configurations. Offset and gain calibration data is stored in SFlash.

Customer software is responsible for reading the correct SFlash data for the desired configuration, interpolating that data between temperatures, and loading the data into hardware registers.

The calibration data requires analog channel usage of ACHAN0: Current with AGC, and ACHAN1: Voltage and diagnostic. For more information on the following registers, see the *PSoC™ 4 HV PA Registers TRM.*

22.2.7.1.1 ACHAN0: Current with AGC calibration procedure

1. Read the SFlash data of offset scaler value from:

Register name:	SFLASH_PACSS_CHAN_OFFSET_SCLR
Description:	Defines the offset scaler for all offset calibration values
Comments:	Defines the offset scaler to be used with the OFFSETADJ for all 2TEMP and 3TEMP calibration values.

Bits	Name	Description
7:0	OFFSETSCLR	Offset scaler

2. Load OFFSETSCLR value into the following register.

Register name:	PACSS_DCHAN0_OFST_COR
Description:	Offset Correction register
Comments:	–

Bits	Name	SFlash Data	Description
19:16	OCOR_SCLR	OFFSETSCLR	Decimator Offset Correction Coefficient Scaler

Note: Only bits 3:0 of OFFSETSCLR in SFLASH should be copied to OCOR_SCLR.

3. Measure junction temperature using the internal temperature sensor.

4. Read the SFlash data of offset and gain adjustment values from:

Register name:	SFLASH_PACSS_CHAN0_2TEMP_TRIMx
Description:	Channel 0 two temperature trim values for software lookup
Comments:	GAIN and OFFSET trim values for PACSS Analog Channel 0. Two of these registers are used to create a 64-bit structure for a single gain setting. The structure contains trims at Tjunction = –40°C (lower word) and 150°C (upper word). The actual trim should be a linear interpolation of the provided trims, using the measured junction temperature. The 64-bit structure is repeated for the following gain values: Index - Gain Value 0 - 1 1 - 2 2 - 4 3 - 8 4 - 16 5 - 32 6 - 64 7 - 128 8 - 256 9 - 512

Bits	Name	Description
15:0	OFFSETADJ	OFFSET adjustment value (signed)
31:16	GAINADJ	GAIN adjustment value (unsigned)

Calibration data of SFLASH_PACSS_CHAN0_2TEMP_TRIMx are provided for two temperatures and ten conditions.
- Temperature: –40°C and 150°C
- Gain: 1, 2, 4, 8, 16, 32, 64, 128, 256, and 512

5. Calculate OFFSETADJ and GAINADJ (except OFFSETSCLR) for all settings by linear interpolation.
6. Load calculated OFFSETADJ and GAINADJ values into the following registers. Note that the current channel uses automatic gain control circuit (AGC). See **"Offset and gain correction"** on page 296 for more details.

Register name:	PACSS_MMIO_OFST_CORx
Description:	Offset Correction register
Comments:	–

Bits	Name	SFlash Data	Description
15:0	OCOR	OFFSETADJ	Decimator Offset Correction Coefficient

Register name:	PACSS_MMIO_GAIN_CORx
Description:	Gain Correction register
Comments:	–

Bits	Name	SFlash Data	Description
15:0	GCOR	GAINADJ	Decimator Gain Correction Coefficient

7. Measure junction temperature again using the internal temperature sensor.
8. If junction temperature has changed (for example, by 5°C from the previous temperature), loop steps 4 to 7, to update the Offset and Gain adjustment values.

Note: When ACHAN0 takes a diagnostic measurement of the VSENSE or VDIAG inputs, use the following formula:

- calculated_gcor_hvch0 = gcor_ch0 × gcor_hvch1 / gcor_ch1
- calculated_ocor_hvch0 = –gcor_ch1 × (ocor_ch1 – ocor_hvch1) / gcor_ch0 + ocor_ch0

where,

gcor_ch0 = ACHAN0 GCOR for 1X Gain

gcor_ch1 = ACHAN1 GCOR for 1X Gain

gcor_hvch1 = ACHAN1 GCOR for HV Input

ocor_ch0 = ACHAN0 OCOR for 1X Gain

ocor_ch1 = ACHAN1 OCOR for 1X Gain

ocor_hvch1 = ACHAN1 OCOR for HV Input

* Use the /16 or /24 version of *hvch1 to match the setting to be used for ACHAN 0

22.2.7.1.2 ACHAN1: Voltage and diagnostic calibration procedure

1. Read the SFlash data of offset scaler value from:

Register name:	SFLASH_PACSS_CHAN_OFFSET_SCLR
Description:	Defines the offset scaler for all offset calibration values
Comments:	Defines the offset scaler to be used with the OFFSETADJ for all 2TEMP and 3TEMP calibration values.

Bits	Name	Description
7:0	OFFSETSCLR	Offset scaler

2. Load OFFSETSCLR value into the following registers.

Register name:	PACSS_DCHAN1(and 2)_OFST_COR
Description:	Offset Correction register
Comments:	–

Bits	Name	SFlash Data	Description
19:16	OCOR_SCLR	OFFSETSCLR	Decimator Offset Correction Coefficient Scaler

Note: Only bits 3:0 of OFFSETSCLR in SFLASH should be copied to OCOR_SCLR.

3. Measure junction temperature using the internal temperature sensor.

4. Read the voltage SFlash data from:

Register name:	SFLASH_PACSS_CHAN1_3TEMP_TRIMx
Description:	Channel 1 three temperature trim values for software lookup
Comments:	GAIN and OFFSET trim values for PACSS Analog Channel 1. Three of these registers are used to create a 96-bit structure for a single gain setting. The structure contains trims at Tjunction = –40°C (lower word), CENTERTEMP (middle word), and 150°C (upper word). The actual trim should be a linear interpolation of the provided trims, using the measured junction temperature. The 96-bit structure is repeated for the following input and gain values: Index - Definition 0 - VSENSE/16 1 - VSENSE/24 2 - VDIAG/16 3 - VDIAG/24.

Bits	Name	Description
15:0	OFFSETADJ	OFFSET adjustment value (signed)
31:16	GAINADJ	GAIN adjustment value (unsigned)

Calibration data of SFLASH_PACSS_CHAN1_3TEMP_TRIMx are provided for three temperatures and four conditions
- Temperature: –40°C, –25°C, and 150°C
- Attenuation 16 and 24, using PGA Gain = 1 and Modulator Gain = 1
- Inputs VSENSE and VDIAG

5. Read the diagnostic SFlash data from:

Register name:	SFLASH_PACSS_CHAN1_2TEMP_TRIMx
Description:	Channel 1 two temperature trim values for software lookup
Comments:	GAIN and OFFSET trim values for PACSS Analog Channel 1. Two of these registers are used to create a 64-bit structure for a single gain setting. The structure contains trims at Tjunction = –40°C (lower word) and 150°C (upper word). The actual trim should be a linear interpolation of the provided trims, using the measured junction temperature. The 64 bit structure is repeated for the following gain values: Index - Gain Value 0 - 0.5 1 - 1 2 - 2 3 - 4 4 - 8 5 - 16 6 - 32 7 - 64 8 - 128 9 - 256 10 - 512

Bits	Name	Description
15:0	OFFSETADJ	OFFSET adjustment value
31:16	GAINADJ	GAIN adjustment value

Calibration data of PACSS_CHAN1_2TEMP_TRIMx are provided for two temperatures and eleven conditions.
- Temperature: –40°C and 150°C
- Gain: 0.5, 1, 2, 4, 8, 16, 32, 64, 128, 256, and 512

6. Calculate OFFSETADJ and GAINADJ (except OFFSETSCLR) for all settings by linear interpolation.

7. Load calculated OFFSETADJ and GAINADJ values into the following registers: Note that the voltage/diagnostic channel uses direct control of gain (for example, voltage = DCHAN1, Diagnostic channel = DCHAN2).

Register name:	PACSS_DCHAN1(and 2)_OFST_COR
Description:	Offset Correction register
Comments:	–

Bits	Name	SFlash Data	Description
15:0	OCOR	OFFSETADJ	Decimator Offset Correction Coefficient

Register name:	PACSS_DCHAN1/2_GAIN_COR
Description:	Gain Correction register
Comments:	–

Bits	Name	SFlash Data	Description
15:0	GCOR	GAINADJ	Decimator Gain Correction Coefficient

8. Measure junction temperature again using the internal temperature sensor.
9. If junction temperature has changed (for example, by 5°C from the previous temperature), loop steps 4 to 8 to update the Offset and Gain adjustment values.

22.2.8 PACSS diagnostic features

The PACSS Diagnostic Features is divided into two sections:

- Internal temperature calculation
- Shunt resistor open detection of RSx pins

22.2.8.1 Internal temperature calculation

The formula for internal temperature calculation is as follows. For more information about internal temperature, see **"On-die temperature sensor"** on page 327.

Note: While making internal temperature sensor measurements, OCOR and GCOR (offset and gain correction) should be disabled.

$$T = \alpha 2 \times x^2 / 2^{34} + \alpha 1 \times x / 2^9 + \alpha 0 \times 2^{14} \qquad (22.6)$$

where:

x = dVBE + dVBE × (A × ratio + B)

dVBE = ADC(UNIT) – ADC(1) with 29 bit signed result

ratio = ADC(UNIT) / ADC(1) with LOAD_MODE set to "Resistor"

A = –0.0534, B = 0.4804

where (A × ratio + B) is a linear approximation of LN(9)/LN(ratio).

ADC(UNIT) is the measurement with UNIT_MODE and IREF_BIPOLAR_UNIT_MASK applied.

The two measurements used for the 'ratio' can also be used a Diagnostic measurement to prove that the channel is working properly.

a2, a1, and a0 coefficients are read from either SFlash (two circuits, Primary and Alternate, are available for diagnostic).

Register name:	SFLASH_PACSS_DIAG_TEMP_P_CAL_Ax
Description:	Primary temperature calibration data polynomial ax
Comments:	–

Bits	Name	Description
15:0	Ax	Temperature calculation polynomial constant coefficient

Register name:	SFLASH_PACSS_DIAG_TEMP_A_CAL_Ax
Description:	Alternate temperature calibration data polynomial ax
Comments:	–

Bits	Name	Description
15:0	Ax	Temperature calculation polynomial constant coefficient

The following is optimal order of calculation. This uses a 29-bit signed result size.

- Optional order of calculation

1. x^2 → shift right 34 bits → multiply by a2
2. x → shift right 9 bits → multiply by a1
3. a0 → shift left 14 bits → sum with above

For fixed-point calculation, it can also be expressed as the above formula divided by 2^{20}.

$$T = (\alpha2 \times x^2 / 2^{34} + \alpha1 \times x / 2^9 + \alpha0 \times 2^{14}) / 2^{20} \tag{22.7}$$

The following defines the setup for the two temperature measurements.

First measurement:

	Reading SFlash Data			**Measurement Setup to Control Register**	
Register Name	SFLASH_PACSS_DIAG_TEMP_P(or A)_CAL_S			PACSS_MMIO_TMPS_CTL	
Bits	8:0	IREF_1_MASK	→	8:0	IREF_UNIT
	18:10	BIPOLAR_1_MASK	→	20:12	BIPOLAR_UNIT
	30	IREF_SEL	→	11	IREF_SEL
	31	LOAD_MODE	→	25:24	LOAD_MODE

Second measurement:

UNIT_MODE and IREF_BIPOLAR_UNIT_MASK define change for the second measurement. For more information, see the PSoC™ 4 HV PA *Registers TRM.*

22.2.8.2 Shunt resistor open detection of RSx pins

The current measurement channel detects disconnect shunt connections by switching internal pull-up resistors to the shunt pins. If the resistance between the RSHx or RSLx to ground is more than 1000 Ω, an open pin fault can be detected. **Figure 22-40** shows the shunt resistor open detection diagram.

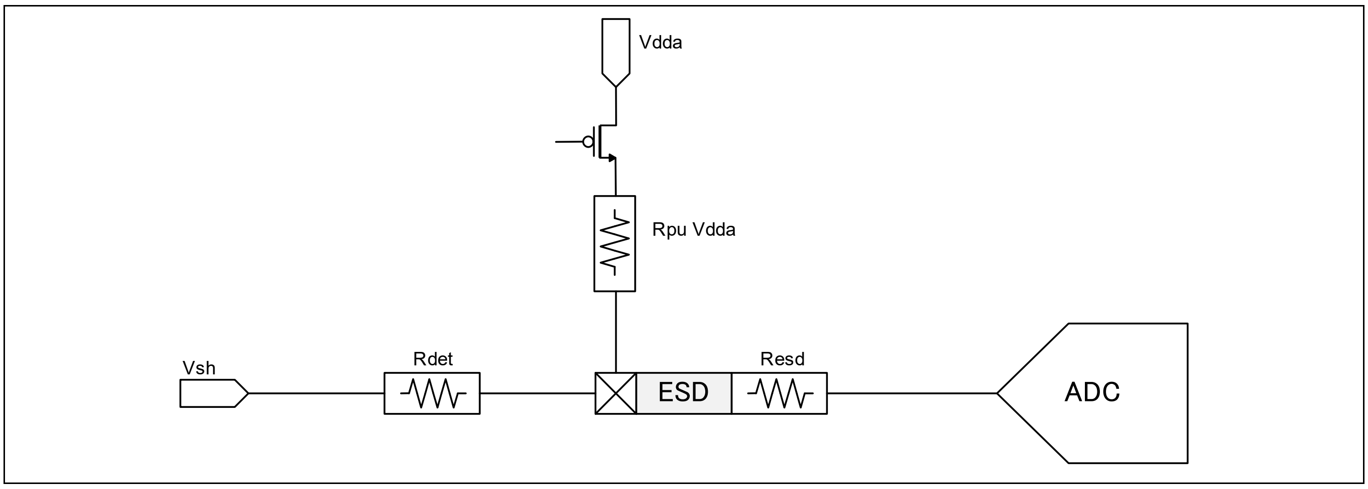


Figure 22-40. Shunt resistor open detection diagram

- Detection criteria:
  - Vadc_th must be defined to separate “good” (lower voltage) from “open” (higher voltage)
  - Rdet < 250 Ω should always return “good” status
  - Rdet > 1000 Ω should always return “open” status

Note that the following components have variation:

- Vsh: –0.3 V to 0.3 V
- Vdda: 2.7 V to 3.6 V
- Rpu: 4000 Ω nominal, ±40%

The following shows algorithm of RS* Open Detection.

1. Enable the pull-up and measure the RS* pad with 0.5x gain: V_{PU}
2. If $V_{PU} > 1.0$ V then Fail: the shunt connection is too high
3. Disable the pull-up and measure the RS* pad with 0.5x gain: V_{SH}
4. Read R_{PU} from SFLASH_PACSS_DIAG_RSxx_PURES, multiply by 32, and interpolate for temperature.

Register name:	SFLASH_PACSS_DIAG_RSxx_PURES
Description:	Measured RSxx pull-up resistor value for shunt open testing
Comments:	Resistance of pull-up resistor, measured during manufacturing. This resistance can be used to reduce the variation of the open-shunt diagnostic test.

Bits	Name	Description
7:0	RES_COLD	Resistance/32 in Ohms, measured at –40°C
15:8	RES_HOT	Resistance/32 in Ohms, measured at 150°C

1. If available, use a measured voltage for V_{DD}, otherwise use 3.3 V.
2. Calculate

$$V_{ADC\_TH} = V_{SH} + (V_{DD} - V_{SH}) \times \frac{600}{(600 + R_{PU})}$$

3. If $V_{PU} > V_{ADC\_TH}$ then Fail: the shunt resistance is above 250 Ω.

22.3 Registers

22.3.1 PACSS Registers

Table 22-19. Precision Analog Channel Subsystem (PACSS) Registers

Offset	Size	Qty.	Width	Name	Description
0x40300000	0x10000	4	–	DCHAN	Digital Channel Configuration Register
0x40340000	0x10000	2	–	ACHAN	Analog Channel Configuration Register
0x403F0000	0x10000	1	–	PACSS_MMIO	PACSS top-level MMIO Register

Definition - x: Channel number, y: Structure number

- Register protection: Register protection are for the critical registers. These registers are protected using a "magic key" written to the PACSS_MMIO_REG_PROT.MAGIC register. After the correct key is written, users can update these critical registers. These are non-retention registers that reset in Deep Sleep power mode.
- How It Works: Setting PACSS_MMIO_REG_PROT register to 0xF08169E7 unlocks access to lock-protected registers. These protected registers cannot be written into unless this value is written into the register. Writing a value other than the magic key will disable access to the registers. The register POR value is set to "unlock" the register access.

The following registers are locked:

- PACSS_MMIO_TRIM_REGL0_CTL
- PACSS_MMIO_TRIM_HPBGRy_CTL
- PACSS_MMIO_TRIM_AREFy_CTL
- PACSS_ACHANx_TRIM_PGAy_CTL
- PACSS_ACHANx_TRIM_BUFy_CTL

22.3.1.1 PACSS_MMIO

The global configuration (pacss_mmio) applies to the entire system. This contains the configuration for the sequencers, the temperature sensor, the AGC, and the logic that muxes data between dchan and achan slices.

Table 22-20. Details

Name	Offset	Qty.	Width	Description
PACSS_MMIO_PACSS_CTL	0x0000	1	32	Precision Analog Channel Control Register
PACSS_MMIO_AGC_CTL0	0x0004	1	32	Auto-Gain Correction Control 0 Register
PACSS_MMIO_AGC_CTL1	0x0008	1	32	Auto-Gain Correction Control 1 Register
PACSS_MMIO_AGC_CTL2	0x000C	1	32	Auto-Gain Correction Control 2 Register
PACSS_MMIO_AGC_CTL3	0x0010	1	32	Auto-Gain Correction Control 3 Register
PACSS_MMIO_TMPS_CTL	0x0020	1	32	Temperature Sensor Control Register
PACSS_MMIO_VPOS_PUMP_CTL	0x0024	1	32	Positive Pump Control
PACSS_MMIO_START	0x0030	1	32	Start Conversion Register
PACSS_MMIO_INTR	0x0040	1	32	System Interrupt Request Register
PACSS_MMIO_INTR_SET	0x0044	1	32	System Interrupt Set Request Register
PACSS_MMIO_INTR_MASK	0x0048	1	32	System Interrupt Mask Register
PACSS_MMIO_INTR_MASKED	0x004C	1	32	System Interrupt Masked Request Register
PACSS_MMIO_INTR_CAUSE	0x0050	1	32	Interrupt Cause Register
PACSS_MMIO_STATUS	0x0060	1	32	Status Register
PACSS_MMIO_STATUS1	0x0064	1	32	Status Register 1
PACSS_MMIO_GAIN_CFGy0	0x01y0	10	32	Gain Configuration Register 0
PACSS_MMIO_GAIN_CFGy1	0x01y4	10	32	Gain Configuration Register 1
PACSS_MMIO_GAIN_OFST_CORy	0x01y8	10	32	Offset Correction Register
PACSS_MMIO_GAIN_GAIN_CORy	0x01yC	10	32	Gain Correction Register
PACSS_MMIO_HPBGR_CTL	0x0500	1	32	High Precision Bandgap Reference Control Register
PACSS_MMIO_HPBGR_DFT_CTL	0x0510	1	32	High Precision Bandgap Reference DFT Control Register
PACSS_MMIO_AREF_CTL	0x0600	1	32	Analog Reference Control Register
PACSS_MMIO_AREF_DFT_CTL	0x0610	1	32	Analog Reference DFT Control Register
PACSS_MMIO_REG_PROT	0x0700	1	32	Register Protection Register
PACSS_MMIO_PDFT_CTL	0x0800	1	32	PACSS DFT Control Register
PACSS_MMIO_TRIM_REGL0_CTL	0xFF00	1	32	Regulator (LDO) Trim 0 Register (lock protected)
PACSS_MMIO_TRIM_HPBGRy_CTL	0xFF04	5	32	High Precision Bandgap Reference Trim Control y (lock protected)
PACSS_MMIO_TRIM_AREFy_CTL	0xFF18	7	32	Analog Reference Trim Control 0 Register (lock protected)

22.3.1.2 DCHAN

The digital channel (dchan) contains the sample configuration data that will be sequenced, such as the pin to be sampled and the filtering and post processing options. Each dchan slice contains a decimator, FIR filter (optional), a post processing block, and data storage. Each PACSS can have up to four dchan slices. The default configuration of the digital data system will instance two dchans with a FIR filter and two dchans without one; therefore, two of the FIR filters are optional.

Table 22-21. Details

Name	Offset	Qty.	Width	Description
PACSS_DCHANx_DCHAN_CTL	0x0000	1	32	Digital Channel Control Register
PACSS_DCHANx_SMP_CTL	0x0004	1	32	Sample Control Register
PACSS_DCHANx_SMP_REF_CTL	0x0008	1	32	Sample Reference Control Register
PACSS_DCHANx_DEC_CTL	0x000C	1	32	Decimator Control Register
PACSS_DCHANx_PP_CTL	0x0010	1	32	Post Processing Control Register
PACSS_DCHANx_OFST_COR	0x0014	1	32	Offset Correction Register
PACSS_DCHANx_GAIN_COR	0x0018	1	32	Gain Correction Register
PACSS_DCHANx_RANGE_LOW	0x001C	1	32	Range Detect Low Value Register
PACSS_DCHANx_RANGE_HIGH	0x0020	1	32	Range Detect High Value Register
PACSS_DCHANx_PGA_GAIN_CTL	0x0024	1	32	PGA Gain Control Register
PACSS_DCHANx_CAP_CFG0	0x0028	1	32	Capacitor Configuration 0 Register
PACSS_DCHANx_CAP_CFG1	0x002C	1	32	Capacitor Configuration 1 Register
PACSS_DCHANx_CAP_CFG2	0x0030	1	32	Capacitor Configuration 2 Register
PACSS_DCHANx_ACC_THRESH	0x0034	1	32	Accumulated DataThreshold Register
PACSS_DCHANx_RESULT	0x0040	1	32	Channel Result Data Register
PACSS_DCHANx_RESULT_TAG	0x0044	1	32	Result Tag Register
PACSS_DCHANx_ACC_RESULT	0x0048	1	32	Channel Accumulated Result Data Register
PACSS_DCHANx_INTR	0x0050	1	32	Interrupt Request Register
PACSS_DCHANx_INTR_SET	0x0054	1	32	Interrupt Set Request Register
PACSS_DCHANx_INTR_MASK	0x0058	1	32	Interrupt Mask Register
PACSS_DCHANx_INTR_MASKED	0x005C	1	32	Interrupt Masked Request Register
PACSS_DCHANx_STATUS	0x0060	1	32	Channel Status Register
PACSS_DCHANx_FCFGy_COEF	0x0yy0	64	32	FIR Configuration Coefficient Register
PACSS_DCHANx_FCFGy_TAP	0x0yy8	64	32	FIR Configuration Tap Value Register

22.3.1.3 ACHAN

The analog channel configuration (achan) is the analog configuration that is static while the sequencer traverses between digital channels. Each achan contains the triggering logic, a sequencer, and data path (DPATH) logic.

Table 22-22. Details

Name	Offset	Qty.	Width	Description
PACSS_ACHANx_ACHAN_CTL	0x0000	1	32	Analog Channel Control Register
PACSS_ACHANx_TR_CTL	0x0004	1	32	Trigger Control for Sequencer Register
PACSS_ACHANx_CHOP_CTL	0x0008	1	32	Chopping Control for Sequencer Register
PACSS_ACHANx_PGA_CTL	0x000C	1	32	Programmable Gain Amplifier Control Register
PACSS_ACHANx_MOD_CTL	0x0010	1	32	Modulator Control Register
PACSS_ACHANx_DPATH_CTL	0x0014	1	32	Datapath Control Register
PACSS_ACHANx_BUF_CTL	0x0018	1	32	Buffer Control Register
PACSS_ACHANx_PUMP_CTL	0x0020	1	32	Pump Control Register
PACSS_ACHANx_REF_CTL	0x0024	1	32	Reference Control Register
PACSS_ACHANx_START	0x0028	1	32	Start Conversion Register
PACSS_ACHANx_INMUX_CTL	0x002C	1	32	INMUX Control Register
PACSS_ACHANx_DEM_CTL	0x0030	1	32	Dynamic Element Matching Control Register
PACSS_ACHANx_SWITCH	0x0034	1	32	Firmware Switch Control 0 Register
PACSS_ACHANx_SWITCH_CLEAR	0x0038	1	32	Firmware Switch Control 0 Clear Register
PACSS_ACHANx_SWITCH_SQ_CTL	0x003C	1	32	Switch Sequencer Control Register
PACSS_ACHANx_SWITCH_STATUS	0x0040	1	32	Switch Status 0 Register
PACSS_ACHANx_DFT_CTL	0x0044	1	32	DFT Control Register
PACSS_ACHANx_DFT_STATUS	0x0048	1	32	DFT Status Register
PACSS_ACHANx_TRIM_PGAy_CTL	0xFF00	5	32	PGA Trim 0 (lock protected) Register
PACSS_ACHANx_TRIM_BUFy_CTL	0xFF14	4	32	Buffer Trim 0 (lock protected) Register

22.3.2 SFlash calibration registers

The SFlash calibration registers are shown here.

Table 22-23. Details

Offset	Register Width	Address Width	Name	Description
0x0FFFE000	8	10	SFLASH.128x4	Supervisory Flash Area

Table 22-24. Details

Name	Offset	Qty.	Width	Description
SFLASH_PACSS_CHAN0_2TEMP_TRIMy	0x280	20	32	Channel 0 two temperature trim values for software lookup.
SFLASH_PACSS_CHAN1_2TEMP_TRIMy	0x2D0	22	32	Channel 1 two temperature trim values for software lookup.
SFLASH_PACSS_CHAN1_3TEMP_TRIMy	0x328	12	32	Channel 1 three temperature trim values for software lookup.
SFLASH_PACSS_CHAN_3TEMP_ CENTERTEMP	0x358	1	8	Defines the center temperature for three temperature trim values.
SFLASH_PACSS_CHAN_OFFSET_SCLR	0x359	1	8	Defines the offset scaler for all offset calibration values.
SFLASH_PACSS_DIAG_RSH_PURES	0x35A	1	16	Measured RSH pull-up resistor value for shunt open testing.
SFLASH_PACSS_DIAG_RSL_PURES	0x35C	1	16	Measured RSL pull-up resistor value for shunt open testing.
SFLASH_PACSS_DIAG_RSH2_PURES	0x35E	1	16	Measured RSH2 pull-up resistor value for shunt open testing.
SFLASH_PACSS_DIAG_RSL2_PURES	0x360	1	16	Measured RSL2 pull-up resistor value for shunt open testing.
SFLASH_PACSS_CHAN1_ETEMP_TRIM	0x362	1	16	Channel 1 gain adjust to be used with external temperature measurements.
SFLASH_PACSS_DIAG_TEMP_P_CAL_S	0x364	1	32	Primary temperature calibration data setup
SFLASH_PACSS_DIAG_TEMP_P_CAL_A2	0x368	1	16	Primary temperature calibration data polynomial a2
SFLASH_PACSS_DIAG_TEMP_P_CAL_A1	0x36A	1	16	Primary temperature calibration data polynomial a1
SFLASH_PACSS_DIAG_TEMP_P_CAL_A0	0x36C	1	16	Primary temperature calibration data polynomial a0
SFLASH_PACSS_DIAG_TEMP_A_CAL_S	0x370	1	32	Alternate temperature calibration data setup
SFLASH_PACSS_DIAG_TEMP_A_CAL_A2	0x374	1	16	Alternate temperature calibration data polynomial a2

Table 22-24. Details (continued)

Name	Offset	Qty.	Width	Description
SFLASH_PACSS_DIAG_TEMP_A_CAL_A1	0x376	1	16	Alternate temperature calibration data polynomial a1
SFLASH_PACSS_DIAG_TEMP_A_CAL_A0	0x378	1	16	Alternate temperature calibration data polynomial a0

22.3.3 Peripheral interconnect trigger group control registers

Table 22-25. Details

Register	Name	Description
PERI_TR_CTL	Trigger Control Register	This register provides software control over trigger activation.
PERI_TR_GROUP2_TR_OUT_CTLx	Trigger Control Register	This register specifies the input trigger for a specific output trigger in trigger group2.

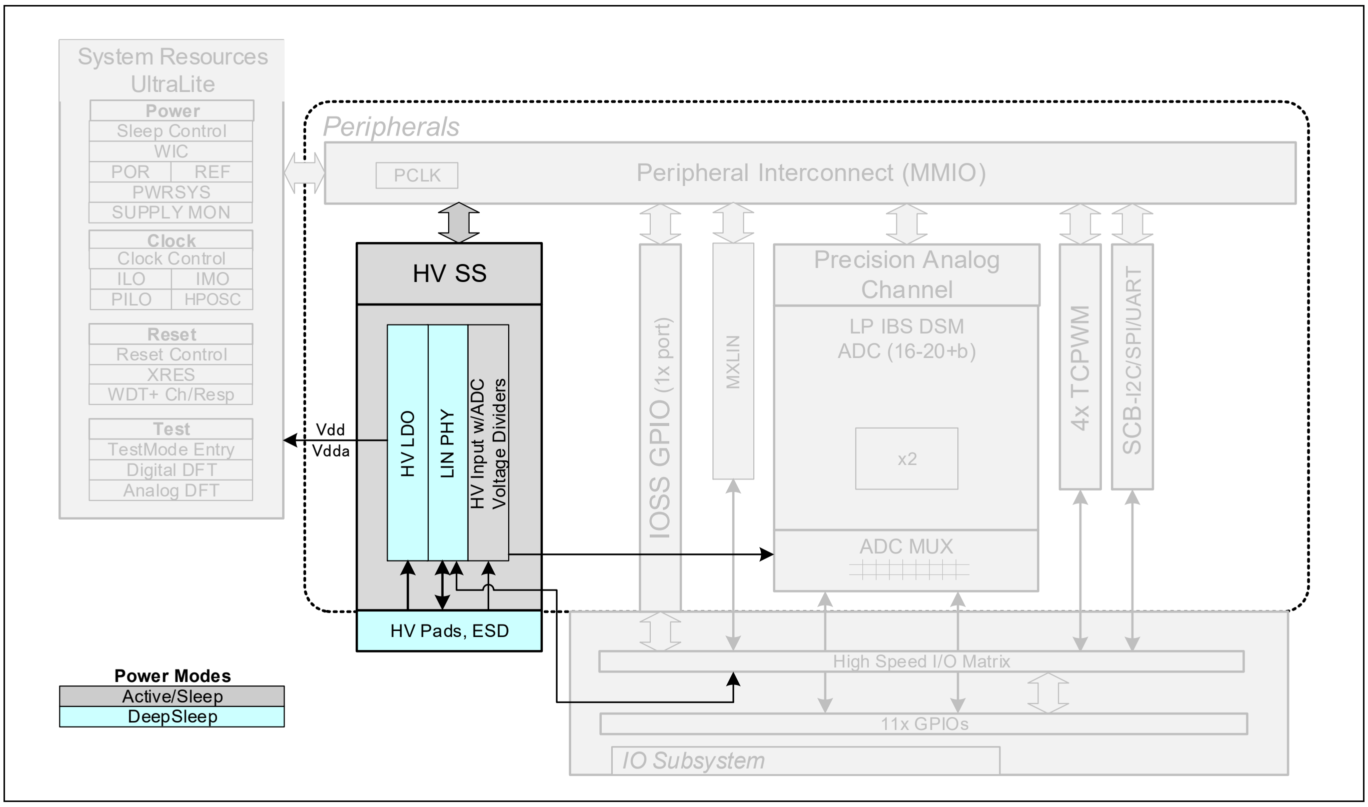
Section F: High-Voltage System

This section encompasses the following chapter:

- **"High-voltage subsystem"** on page 345

Top Level Architecture

High-Voltage Subsystem Block Diagram



23 High-voltage subsystem

The PSoC™ 4 HV PA High-Voltage Subsystem (HVSS) contains a series of analog circuits used in battery monitoring and management applications. The applications are directly connected to an automotive battery and operates up to 42 V. The HVSS has the following features:

- HVSS circuits operate directly off 12-V/24-V automotive battery (tolerates up to 42 V)
- A linear regulator powered by automotive battery voltage producing 3.3 V (HVREG)
  - Output voltage accuracy: 2.7 V to 3.6 V
  - Input voltage range: 3.6 V to 42 V
  - Output current capability: 30 mA (for core, GPIO, and external loads), 10 mA (for GPIO and external loads)
  - Low current consumption for always on: maximum 20 µA
  - Power good function
- A LIN physical interface transceiver (LIN PHY)
  - Data rates up to 20 Kbps with high EM noise immunity
  - Positive/negative DC tolerance for LIN pin: –27 V to 42 V
  - Sleep mode current consumption: $I_{VBAT} + I_{VDDD} = 10$ µA max
  - Programmable slew rate control: 1.0 V/1.5 V/2.0 V/µs at $V_{BAT} = 12$ V
  - Dominant timeout and LIN wakeup interrupt timers
- A set of input voltage dividers (VDIVIDER) providing attenuated signals to ADCs
  - Dividers attenuate input by factor of 16x or 24x
  - 1.2-V full-scale outputs from input of 19.2 V (16x) or 28.8 V (24x)
- An AHB bus interface and control/status registers
- Interrupt logic

23.1 Block diagram

The HVSS has three public cells: HVREG, LIN PHY, and VDIVIDER. **Figure 23-1** shows the block diagram for the HVSS.

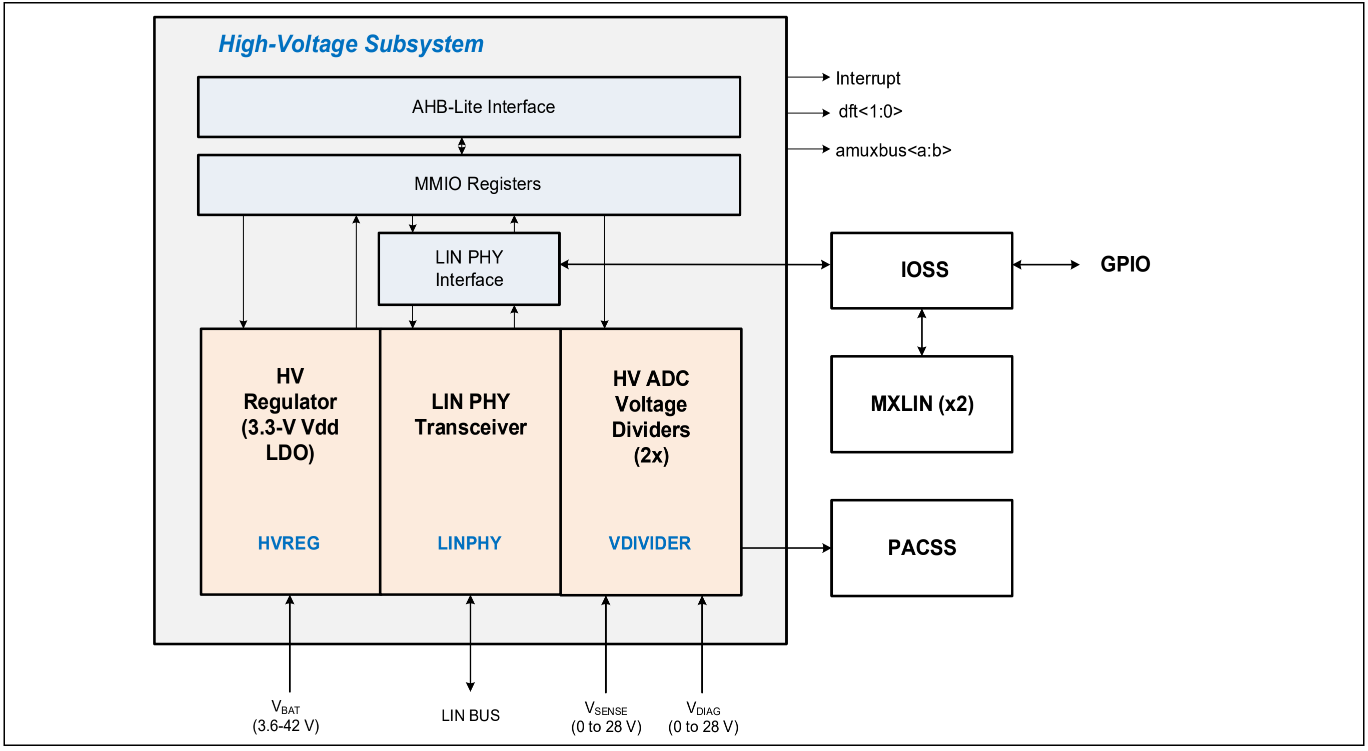


Figure 23-1. Block diagram

23.1.1 HVREG

The HVREG is the high-voltage regulator with self-start from the battery without a reference voltage; it provides 3.3 V to V_{DDD} and V_{DDA} as the chip supply. It also contains a power good function and Zener diode on the output pin for over-voltage protection. The HVREG operates with stability against automotive battery voltage transient events, keeping the output voltage between 2.7 V to 3.6 V even if the V_{BAT} pin drops down to 3.6 V in cold cranking or rises up to 42 V in a load dump surge.

23.1.2 LIN PHY

The LIN PHY is the interface between the Local Interconnect Network (LIN) protocol controller and the physical bus in a LIN, which meets the requirements of LIN standard 2.2A and is downward-compatible with LIN 2.0. It supports data rates of 1 to 20 Kbps. Non-LIN fast slew rates are available, providing 100 Kbps data rates for fast downloads; this is used for factory and field flash program updates using the LIN pin.

23.1.3 VDIVIDER

The VDIVIDER is a voltage divider used on the VSENSE and VDIAG pins to scale battery voltage to levels compatible with on-chip ADCs; this ensures that battery voltages are measured. In typical battery monitoring and management applications, the VSENSE input is normally connected directly to the battery with a series 2.2-kΩ resistor to measure battery voltage. VDIAG can be used to measure voltage at other locations such as the ECU or other loads where monitoring is desired. The external 2.2-kΩ resistors in series with voltage sources limit current during ESD and transient voltage events.

23.2 How it works

The HVSS includes circuitry that interfaces directly with automotive battery power and signals (nominal 12 V to 28 V, tolerates up to 42 V). The HVSS is configured and controlled by an AHB-Lite interface using memory mapped I/O registers. **Figure 23-2** shows an architectural diagram of the HVSS with key off-chip components.

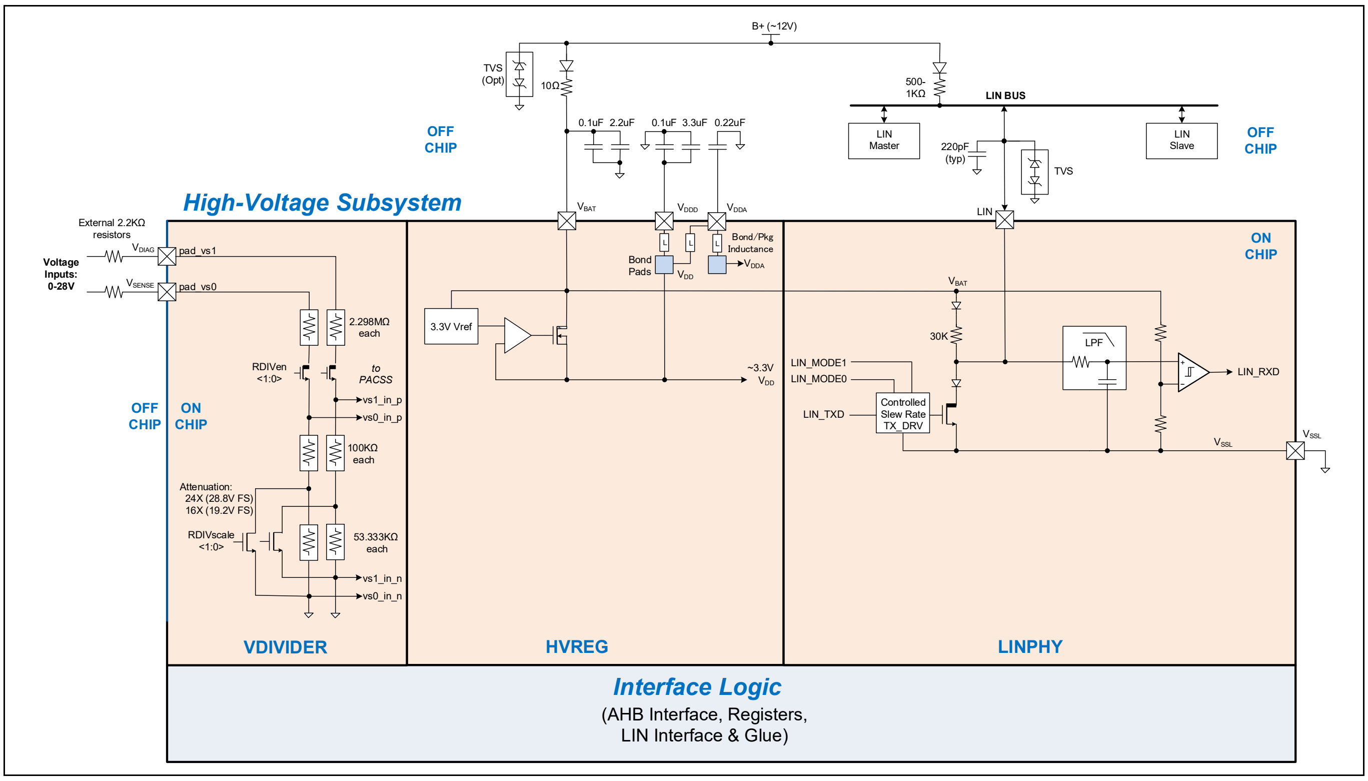


Figure 23-2. HVSS architecture diagram with off-chip components

On the left in **Figure 23-2**, a voltage divider attenuates input signals by a factor of 16x or 24x, providing 1.2-V full-scale outputs from input voltages of 19.2 V (16x) or 28.8 V (24x). The voltage divider can be disabled to reduce input current when measurements are not needed.

The HVREG provides nominal 3.3 V from a 3.6 V to 42 V power supply, which is usually a car battery. This block is always powered and is designed to consume low-power quiescent power (about 15 µA).

The block on the right is a LIN PHY, which is a transceiver that connects to an external LIN bus. The LIN driver provides a pull-down to signal a dominant state and relies on pull-up resistors to signal a recessive state. The driver includes slew control and edge rounding to minimize EMI generation and has internal filtering to minimize EMI susceptibility. The receiver translates the state of the LIN bus to an on-chip logic signal and includes filtering to minimize noise susceptibility. Normally, the LIN receiver is ON in Deep Sleep mode to listen for LIN communication. The HVSS system interfaces these to LIN signaling in the IOSS, or to an alternative path for test purposes.

23.2.1 HVREG

Figure 23-3 shows the HVREG block architecture and function diagram. The HVREG generates a nominal 3.3-V output voltage with loads up to 30 mA from an external supply voltage input of 3.6 V to 42 V. The regulator maintains output voltage from 3 V to 3.6 V when the supply voltage is greater than 4 V, and may drop to 2.7 V when the supply is between 3.6 V to 4 V. In normal operation, the external automotive battery voltage is typically 12 V or higher but can drop as low as 4.5 V during cold cranking. However, voltage at the V_{BAT} pin can go as low as 3.6 V due to a reverse polarity protection diode and an EMI filter resistor.

This regulator provides V_{DDD} (and V_{DDA}) to power the chip, including the SRSS regulators, which then generate V_{CCD} (1.8 V) supplies. This regulator is always powered on. The regulator has one output for both V_{DDD} and V_{DDA}. Bond wire inductance and a dedicated bypass capacitor provide some V_{DDA} filtering.

This regulator has a self-bias circuit and can self-start by connecting the battery to the V_{BAT} pin. The regulator contains a power good function, which outputs a low pg_hv signal (HVSS_HVREG_STATUS.PWR_GOOD) when the V_{DDD} output is less than 2.0 V.

The pg_hv signal indicates "power good", that the V_{DD} supply is stable. The pg_hv is routed to SRSS to optionally provide a chip reset when power is not good. This signal is referred to as "BODHVSS" inside SRSS. HVREG also has a Zener diode on its output pin for functional safety.

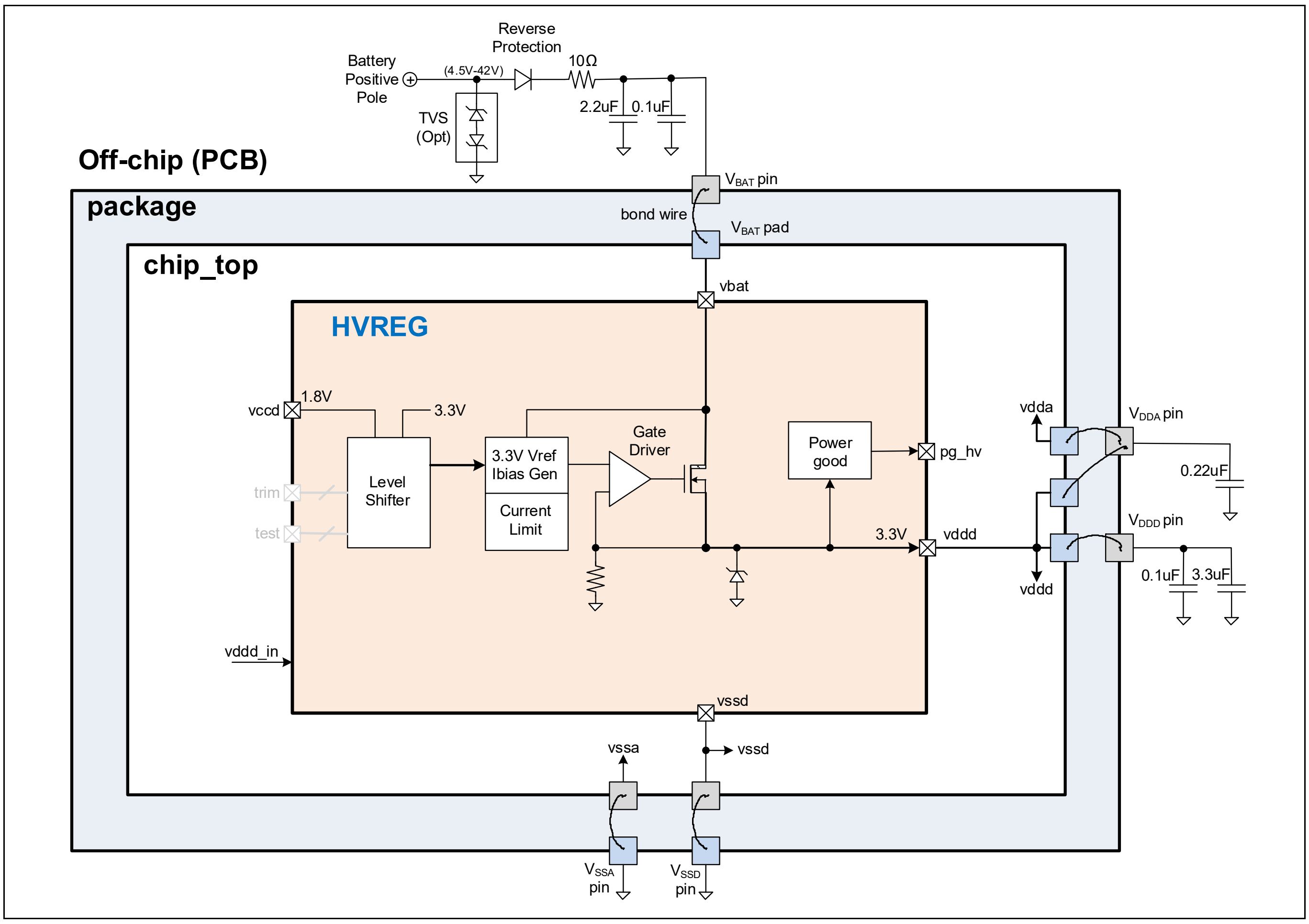


Figure 23-3. HVREG block architecture and function diagram

Startup and shutdown timing diagrams of HVREG are shown in **Figure 23-3**.

HVREG starts up when the battery is connected to V_{BAT}. When V_{DDD} reaches between 2.05 V and 2.7 V, pg_hv becomes "H". HVREG then outputs the proper trimmed voltage. When the battery is removed from the V_{BAT} pin, V_{DDD} shuts down.

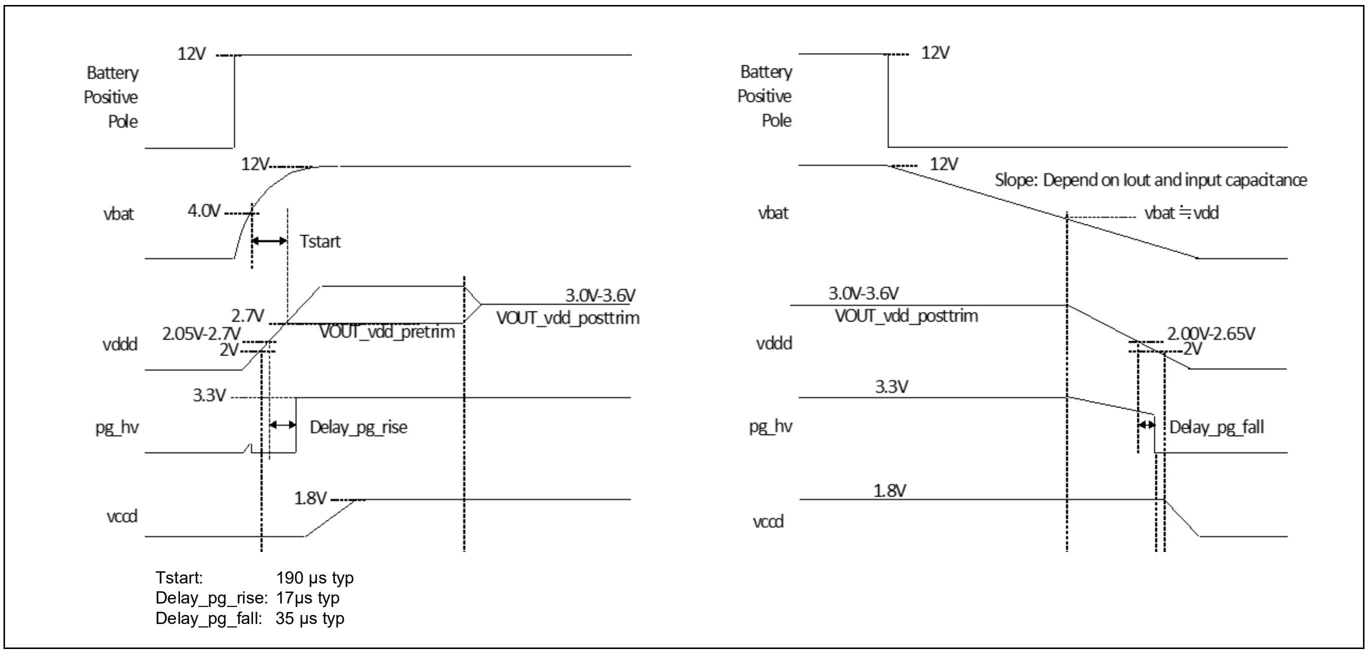


Figure 23-4. HVREG startup and shutdown timing diagrams

23.2.1.1 HVREG input pins

vddd_in is an analog input signal pin to monitor V_{DDD} voltage level by power good circuit (PG). This pin connects to the output V_{DDD} at the appropriate place in the device, allowing for V_{DDD} voltage drop.

23.2.1.2 HVREG output pins

"vddd" is the output pin of this regulator, a nominal 3.3 V regulated voltage with loads up to 30 mA. This output drives the device's V_{DDD} and V_{DDA}, and through the GPIO can drive external loads. The vddd pin is always ON after startup.

pg_hv is the output pin of power good circuit in HVREG (BODHVSS: routed to SRSS as reset source). HVREG monitors the V_{DDD} output voltage net through the vddd_in pin. When V_{DDD} is lower than 2 V, pg_hv provides a secure "L". Hence, this output signal can work as the reset signal of the entire device in the voltage range lower than Vddd = 2.0 V.

23.2.1.3 Reset and initialization

HVREG has no registers and latch circuits to hold states. The block provides power to the rest of the device and thus cannot be reset or disabled. However, HVREG will ignore the input when the pg_hv signals are low.

23.2.1.4 Power mode

After the V_{BAT} pin is connected to the automotive battery, HVREG continues to be powered on with a circuit current consumption of 20 µA or less to provide internal power supplies V_{DDD} and V_{DDA}. Power On/Off depends on the battery connection. It operates with low current consumption of about 15 µA in Deep Sleep mode.

23.2.2 LIN PHY

23.2.2.1 LIN specification overview

Local interconnected network (LIN) is a low-cost serial communication protocol, standardized by the LIN consortium. The main properties of LIN are:

- Single master with one or more slaves
- Speeds up to 20 kbit/sec
- Single-wire implementation
- Low-cost silicon implementation based on common UART/SCI interface hardware or equivalent software/hardware state machine

The physical interface is a single-wire (not including ground) using a physical layer interface (PHY). The LIN PHY has two states: the recessive state where the LIN bus is pulled up close to the vehicle battery voltage (a reverse protection diode causes a small voltage drop) and a dominant state, which is essentially 0 V. **Figure 23-5** shows an overview of a LIN network.

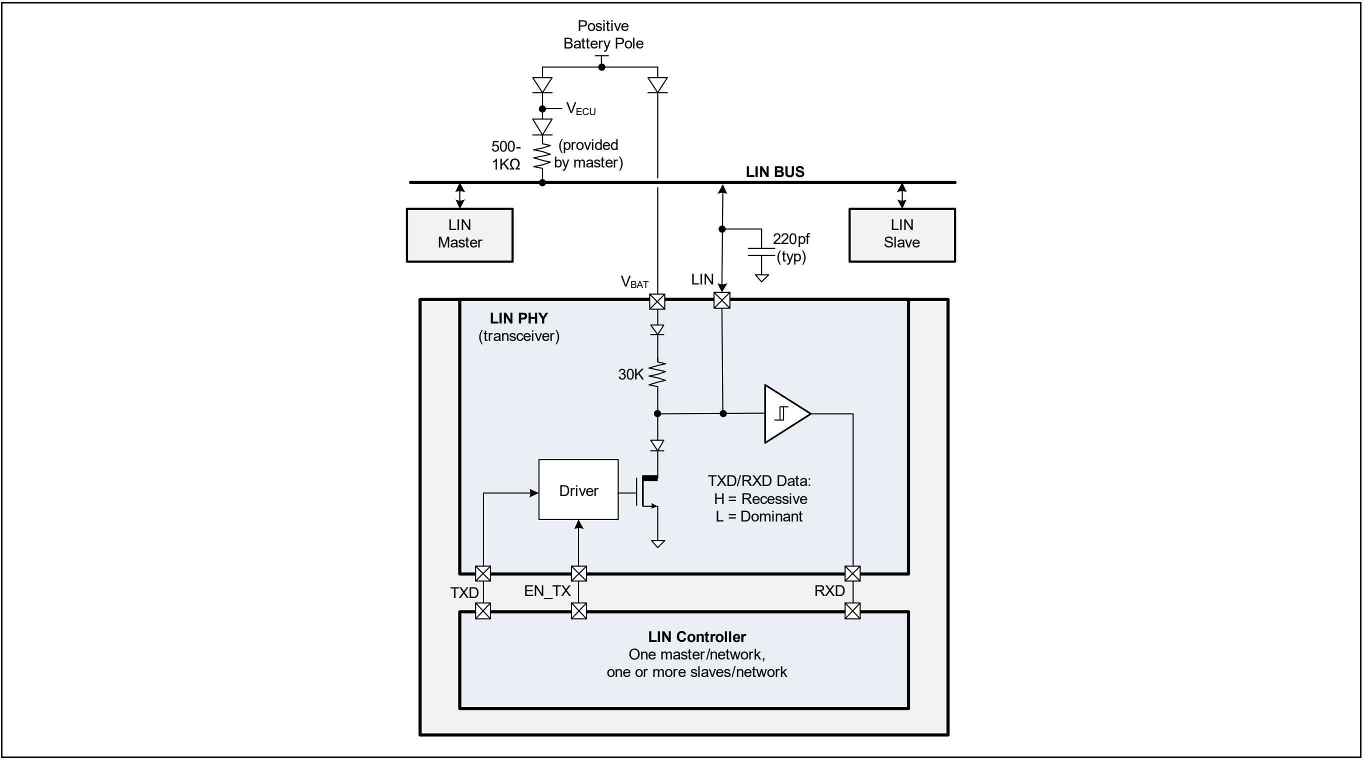


Figure 23-5. Local interconnect network (LIN) overview

Each node on the LIN network uses a LIN PHY (transceiver) to communicate over the network. The LIN physical layer is independent of higher LIN protocol layers including master and slave operation. There are no differences between master and slave transceivers. Each LIN network can have up to 16 nodes. The PHY has a transmitter section, which includes the open-drain pull-down transistor and gate driver circuitry, a receiver, and a small pull-up resistor.

The master node includes a termination resistor that ranges from 500 Ω to 1000 Ω depending on network speed and bus length (up to 40 m). The LIN PHY specification specifies network loads of 1000-Ω pull-up with 1-nF bus load capacitance, 680 Ω with 6.8 nF, and 500 Ω with 10 nF. All nodes include a nominal 30-k nominal pull-up resistor; this is intended to compensate for increasing node capacitance (typically ~270 pF/node) to maintain approximately the same pull-up resistor to load capacitance (R × C) time constant as nodes are added. Typically used network speeds are either 10k or 20k bits per second.

The LIN specification has certain isolation requirements to ensure that the LIN bus is not compromised by certain faults. This includes ensuring that loss of supply voltage or ground of a slave node does not affect LIN communication between other nodes. This requirement introduces reverse protection diodes in series with pull-up resistors and another diode in series with the open-drain pull-down resistor to prevent powering components from the LIN bus and excessively loading the LIN bus and disrupting communication.

23.2.2.2 LIN PHY transceiver

Figure 23-6 shows the LIN PHY transceiver block diagram.

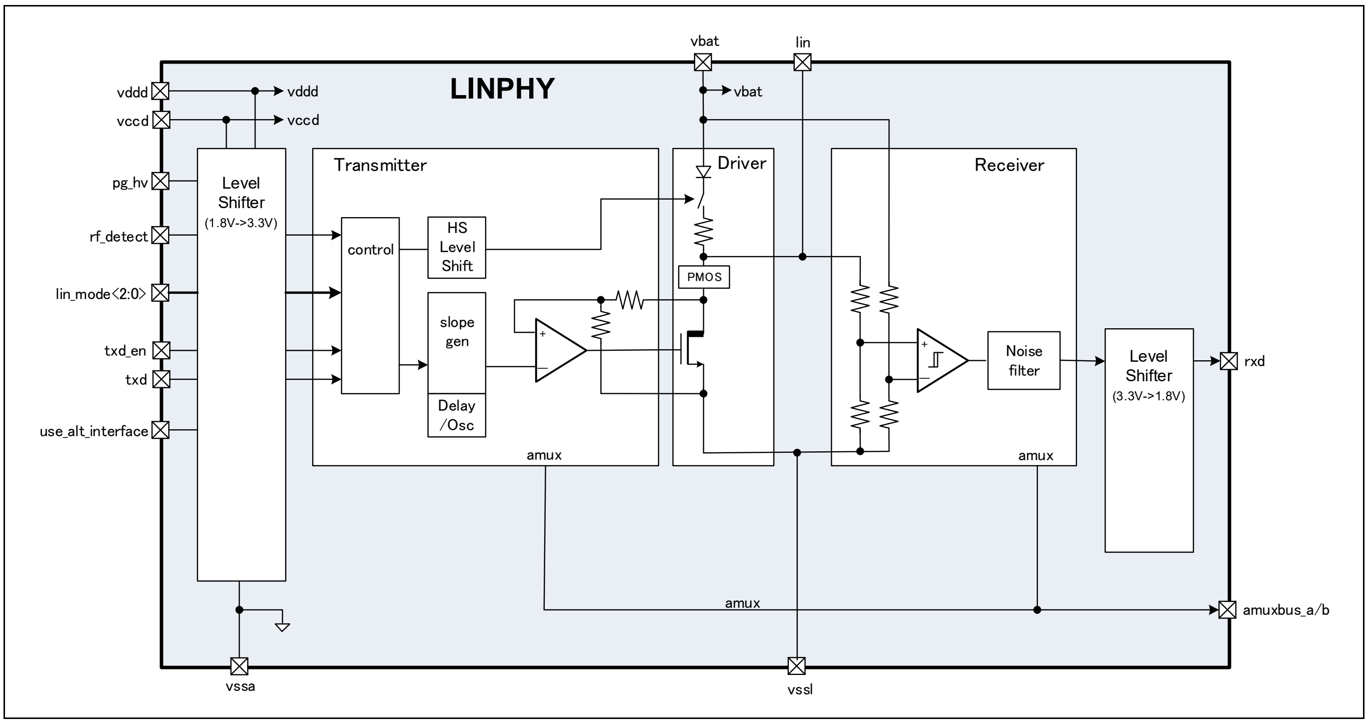


Figure 23-6. LIN PHY block diagram

The LIN driver includes slew control to maintain consistent duty cycle over a range of loads. It also includes turn-on/turn-off control to round edges and minimize EMI. Large amounts of RF energy can be present on the LIN bus; the driver must remain functional in the presence of this energy, which requires special attention to the design of feedback circuits, so they do not saturate in the presence of RF energy.

LIN receiver thresholds are specified in relation to the supply voltage V_{BAT}. A comparator with voltage dividers sets the receiver threshold and scales bus voltages to appropriate levels. Noise filtering is needed to prevent RF energy and brief transients from disturbing communication.

23.2.2.3 Mode transition

Figure 23-7 illustrates the mode transition diagram of the LIN PHY transceiver block. The LIN PHY function modes are controlled by changing HVSS_LIN_CTL.LIN_MODE [2:0] code, as shown in **Table 23-1**.

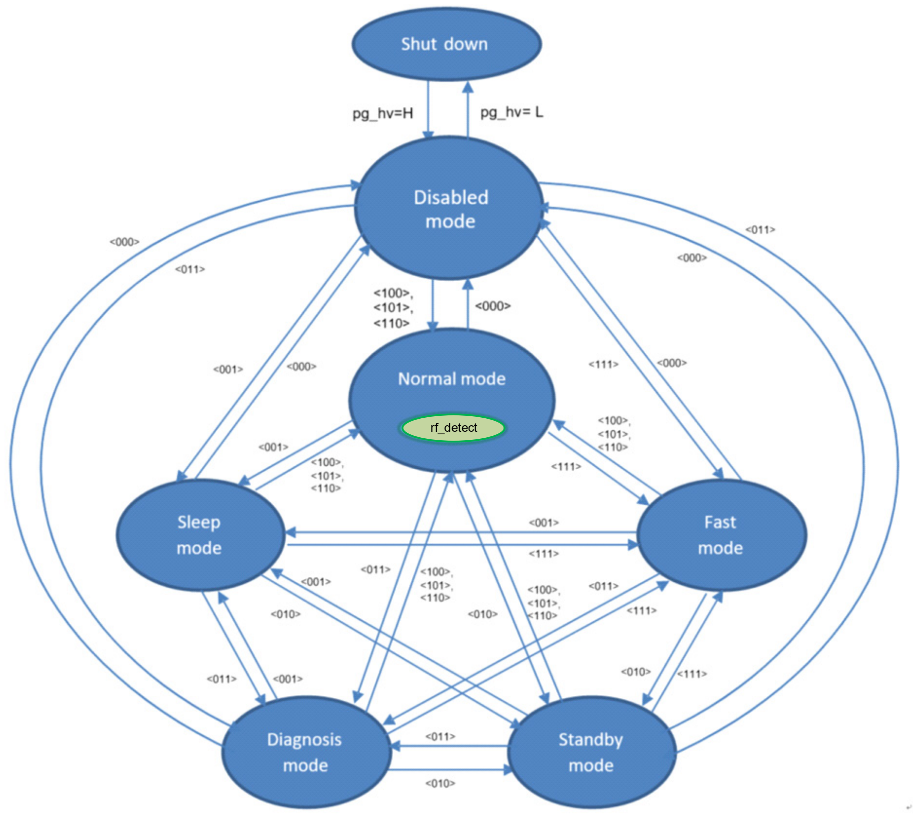


Figure 23-7. LIN PHY mode transition diagram

A brief description of each mode is as follows:

- **Disabled Mode**
  At startup when pg_hv is set, the LIN PHY block will be in the Disabled mode, with HVSS_LIN_CTL.LIN_MODE [2:0] = <000>. In this mode, LIN PHY does not operate and the transmitter connected to LIN bus outputs Hi-Z. Therefore, its consumption current will be zero except for leakage.
- **Normal Mode**
  When HVSS_LIN_CTL.LIN_MODE [2:0] is set to each of <100>, <101>, and <110>, LIN PHY goes into Normal mode. In this mode, LIN PHY works as the interface between the device's LIN protocol controller and the physical bus, which meets the requirements of LIN standard 2.2A and is downward-compatible with the LIN 2.0. Data rates of 10 Kbps and 20 Kbps are supported. Three types of slew rates, 1.0/1.5/2.0 V/µs at V_{BAT} = 12 V, can be set and adjusted according to the LIN specification. The falling slope of a LIN bus waveform can be made more moderate by setting HVSS_LIN_CTL.RF_DETECT = "H" for EM emission reduction.
- **Sleep Mode**
  When HVSS_LIN_CTL.LIN_MODE [2:0] is set to <001>, LIN PHY goes into Sleep mode. In this mode, only the receiver works while the transmitter sleeps and its driver outputs the "recessive" state to the LIN bus. The receiver works with low power consumption. With HVSS timers, the receiver may detect a wake-up signal on the bus.

- **Fast Mode**
  When HVSS_LIN_CTL.LIN_MODE [2:0] is set to <111>, LIN PHY is in Fast mode. This mode works the same as Normal mode except its slew rate is not limited. Therefore, data rates up to 100 Kbps are supported. However, other electrical specifications may not meet the LIN specification. This mode is only for test time reduction (such as downloads) or debugging.
- **Diagnosis Mode**
  When HVSS_LIN_CTL.LIN_MODE [2:0] is set to <011>, LIN PHY goes into Diagnosis mode. This mode works the same as Normal mode except for the control of the transmitter driver output. For txd = H, the operation is the same. For txd = L, the high side switch of the output stage is completely OFF, and low side switch is replaced by a weak 30-kΩ pull-down resistor instead of the main diode switch.
- **Standby Mode**
  When HVSS_LIN_CTL.LIN_MODE [2:0] is set to <010>, LIN PHY goes into Standby mode. In this mode, LIN PHY does not work and the transmitter outputs the “recessive” state regardless of the txd input. This mode is usually not used; it is suitable for quick wake-up, but the current consumption is high.

When HVSS_LIN_CTL.RF_DETECT is set to “H” in Normal mode, the falling slope of the LIN bus waveform will be more moderate for EM emission reduction (see **Figure 23-8**).

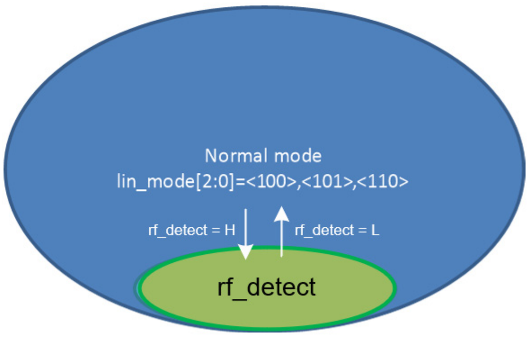


Figure 23-8. HVSS_LIN_CTL.RF_DETECT Mode transition diagram

23.2.2.4 Timers

LIN PHY has a wakeup timer and a dominant timeout fault timer. The wakeup timer can be configured to generate an interrupt when a LIN wakeup is detected. The dominant timeout fault timer inhibits transmission if the TX signal is asserted for too long. This timer can generate an interrupt and transmission is re-enabled by disabling and re-enabling the LIN PHY.

These timers are programmable and are clocked with a low-frequency clock with a nominal 32-kHz frequency.

- **Enable/Disable**
  The timers are enabled through an AHB register write to the HVSS_LIN_TIMER register (FAULT_TIMER_EN and WAKEUP_TIMER_EN bit).
- **Wakeup timer**
  The wakeup timer monitors the incoming LIN receive value to detect a minimum dominant state on the LIN bus, which indicates a wakeup signal to LIN slaves. The standard “wakeup” time is on the order of 150 µs. Because the timer runs on LFCLK, a 32-kHz clock (nominal 30 µs period), the typical period for this timer will be three or four cycles.
  This timer is a down counter, which is initially loaded with the period (HVSS_LIN_TIMER.WAKEUP_TIMER). After it is enabled (HVSS_LIN_TIMER.WAKEUP_TIMER_EN), and when the lin_rxd signal (received LIN data signal) is in the dominant state (0), the timer will down count. If the lin_rxd signal transitions to the recessive state (1) and spans at least one LFCLK edge, this is synchronized and the period is subsequently reloaded to reset the count.
  If the dominant state is continuously asserted for the full Timer Period, the timer will reach terminal count, and the HVSS Interrupt will be asserted.
  This operation will work in Deep Sleep mode, and the output interrupt may be used as a Deep Sleep wakeup source for the device. After it is awake, the interrupt interface is accessible to user code.
  When the timer is disabled by resetting the AHB enable register bit, the timer state will reset.
  Important note: Do not toggle the WAKEUP_TIMER_EN bit when the device comes out of Deep Sleep. It creates a false edge that restarts the wakeup timer when the device goes back into Deep Sleep.
- **Fault timer**
  The fault timer monitors the internal transmit signal to detect if there is a stuck-at fault to the dominant state in the hardware. If this is detected, this function will gate the internal transmit signal to the recessive state.
  The functionality of the fault timer is similar to the wakeup timer. It is a down counter, which is initially loaded with the period (HVSS_LIN_TIMER.FAULT_TIMER). When enabled (HVSS_LIN_TIMER.FAULT_TIMER_EN), down counting will occur when the lin_tx is in the dominant state (0). The period is reloaded for a new count when the transmit data is in the recessive state.
  This operation will work in Deep Sleep mode, and the output interrupt may be used as a Deep Sleep wakeup source for the device. After it is awake, the interrupt interface is accessible to user code.
  When the timer is disabled by resetting the AHB enable register bit, the timer state will reset.
- **Interrupts**
  The terminal count output from the wakeup and/or fault timers sets the interrupt bit. There are four interrupt-related registers: HVSS_LIN_INTR, HVSS_LIN_INTR_SET, HVSS_LIN_INTR_MASK, and HVSS_LIN_INTR_MASKED.

23.2.2.5 LIN PHY input pins

The pg_hv input functions as cell reset signals. When pg_hv is low, the cell is reset. This disables the cell and sets level shifters to their default states. These are V_{DDD}-level signals.

lin_mode<2:0> (HVSS_LIN_CTL.LIN_MODE<2:0>) is an input to change the LIN PHY function mode. These modes are described in **Table 23-1**.

txd is the data input signal, giving the drive state intended at the LIN pin in LIN PHY transmit mode.

txd_en is an input signal to mask the txd signal. When txd_en is "L", the transmitter is controlled so that the txd input is ignored and the LIN pin becomes "recessive" (weak pull-up).

rf_detect (HVSS_LIN_CTL.RF_DETECT) is an input configuration signal to reduce the impact on the LIN driver from incoming EMI. See **Table 23-2** for options. When the rf_detect mode is selected, the falling slope of the LIN bus waveform will be more moderate for lower EMI emission.

use_alt_interface is an input signal to select the primary or alternate interface for Phy. When use_alt_interface is "L", the primary interface is selected (connected to the internal LIN controller through HSIOM). When it is "H", the alternate interface (connected to GPIOs through HSIOM).

23.2.2.6 LIN PHY output pins

"lin" is the input/output pin for LIN PHY transceiver. The block's transmitter can drive this pin, and the receiver uses it as an input. This pin requires positive/negative tolerance from –27 V to 42 V (to account for a "lost ground" mode that can give negative inputs, and battery surge cases for the high positive voltages).

"rxd" is the output of the LIN PHY receiver. The voltage level of this signal is V_{CCD} and reflects the state of the LIN pin.

23.2.2.7 Reset and initialization

LIN PHY has no registers and latch circuits to hold states. However, LIN PHY will ignore low-voltage inputs when pg_hv is low. The HVSS_LIN_CTL.LIN_MODE[2:0] input bus can be used the enable or disable the transceiver (**Table 23-2**).

23.2.2.8 Power modes

LINPHY has three primary power modes (see **Table 23-1**): shutdown mode, active mode, and sleep mode. When HVSS_LIN_CTL.LIN_MODE<2:0> is set to <000>, LINPHY goes into shutdown/disabled mode. In this mode, LINPHY stays shut down with no power consumption. When HVSS_LIN_CTL.LIN_MODE<2:0> is set to <001>, LINPHY goes into sleep mode. In this mode, only the receiver is enabled, listening to LIN pin traffic that may signal a wakeup event. Current consumption is about 10 µA in this mode. When HVSS_LIN_CTL.LIN_MODE<2:0> is set to any other setting, the cell is considered to be in active mode.

The standby mode (state 2) is similar to sleep mode, but has a higher receiver power for faster wakeup.

23.2.2.9 Truth Tables

Table 23-1, **Table 23-2**, and **Table 23-3** show the LIN PHY truth tables.

Table 23-1. lin_mode [2:0] Truth Table

Input				**Output (Driver)**				
State No.	**LIN Transceiver Function**	**LIN_MODE [2:0]**	**Receiver**	**Slew Control**	**Slew Rate setting**	**txd = 1**	**txd = 0**	**Pull-up**
0	Disable	000	Off	No	–	Hi-Z	Hi-Z	Off
1	Sleep	001	On (Low Power)	No	–	Recessive	Recessive	On
2	Standby	010	On	No	–	Recessive	Recessive	On
3	Diagnosis	011	On	No	–	Recessive	Low (weak)	On/Off
4	Normal Mode	100	On	Yes	1.0 V/µs	Recessive	Low (dominant)	On
5		101	On	Yes	1.5 V/µs	Recessive	Low (dominant)	On
6		110	On	Yes	2.0 V/µs	Recessive	Low (dominant)	On
7	Fast Mode	111	On	No	–	Recessive	Low (dominant)	On

Note: State No. 5 meets all parameters of LIN spec. State No. 4 or 5 may not meet all parameters of LIN spec at some operating conditions.

Table 23-2. rf_detect Truth Table

Input	**Output**
rf_detect	**LIN Transceiver Function**
“L” (0)	off (default)
“H” (1)	reduce impact on LIN driver from incoming EMI

Table 23-3. use_alt_interface Truth Table

Input	**Output**
use_alt_interface	**LIN Transceiver Function**
“L” (0)	Primary interface (connected to internal LIN controller through HSIOM)
“H” (1)	Alternate interface (connected to GPIOs through HSIOM)

23.2.3 VDIVIDER (high-voltage divider)

VDIVIDER is used to scale high-voltage analog inputs down to levels compatible with the ADC full-scale input voltage (1.2 V). These programmable dividers support two voltage scales - 16x (19.2-V full scale) or 24x (28.8-V full scale). Each divider can be independently enabled or disabled. The divider generates a single-ended differential signal for the ADC with the bottom of the divider referenced to vdiv_ret.

Figure 23-9 shows a block diagram of the VDIVIDER.

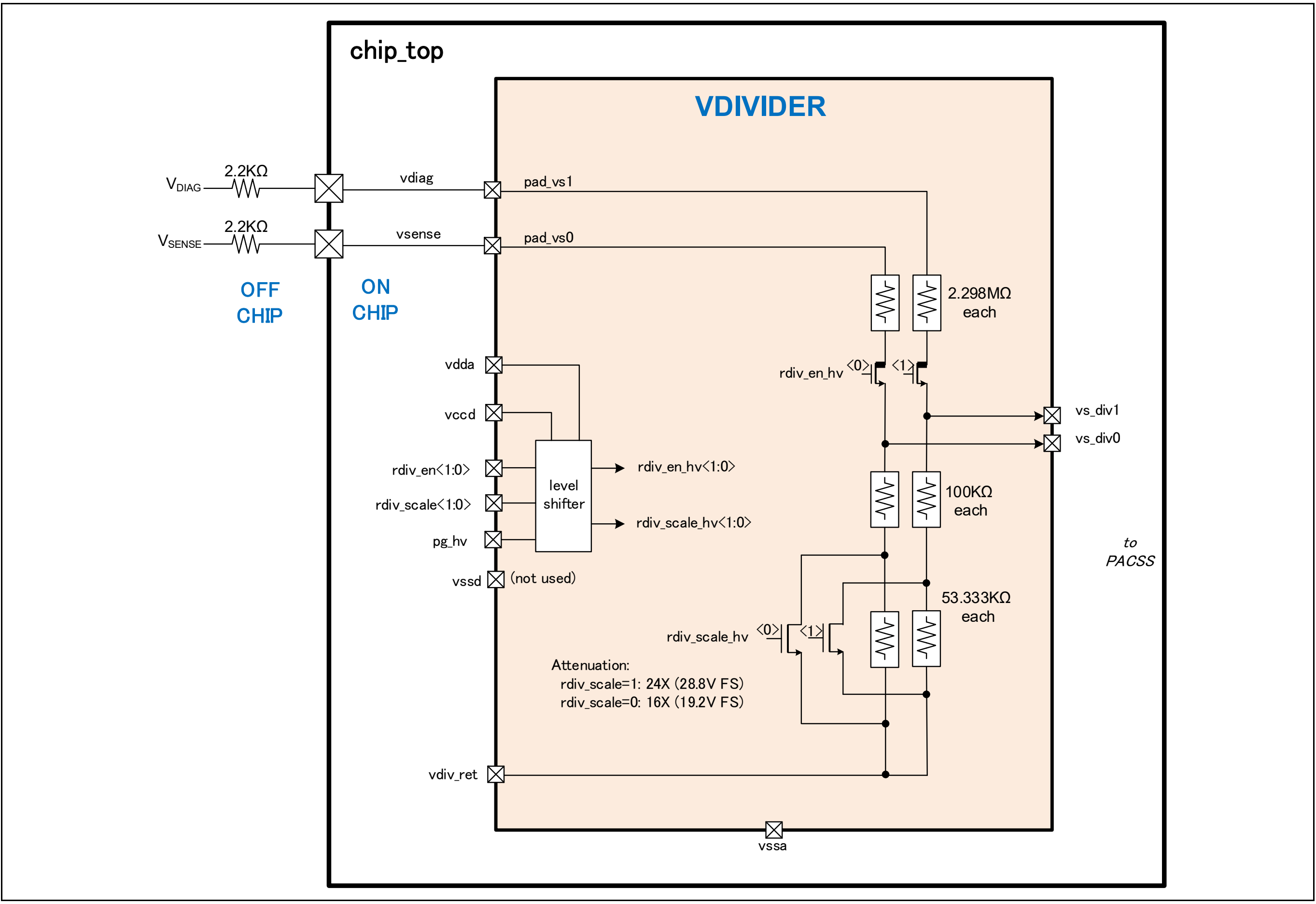


Figure 23-9. VDIVIDER block diagram

VDIVIDER has two voltage attenuators for the two inputs vsense and vdiag. The two dividers can be enabled/disabled independently with the HVSS_RDIV_CTL.RDIV_EN<1:0> configuration inputs. Disabling a divider stops current drawn from its input (other than leakage).

VDIVDER can change its attenuator ratio to either 1/16 or 1/24 with the HVSS_RDIV_CTL.RDIV_SCALE<1:0> configuration inputs.

23.2.3.1 VDIVIDER input pins

pg_hv is a reset signal input pin. While this pin is “L”, the level shift circuit is in reset. When pg_hv is “L”, vs_div0/vs_div1 output “L” regardless with rdiv_en<1:0> voltage level. The voltage level of this bus signals is V_{DDD}.

rdiv_en<1:0> (HVSS_RDIV_CTL.RDIV_EN<1:0>) is an input bus for enabling the resistor dividers of the vsense and vdiag inputs. When either signal is set to “L”, the corresponding resistor ladder is opened and its output becomes low. The voltage level of this bus signals is V_{CCD}.

rdiv_scale<1:0> (HVSS_RDIV_CTL.RDIV_SCALE<1:0>) is an input bus to set the voltage attenuator ratio to either 1/16 or 1/24. **Table 23-4** shows the truth table. The voltage level of this bus is V_{CCD}.

23.2.3.2 VDIVIDER output pins

vs_div0/vs_div1 are output pins of the divided vsense/vdiag signals. The voltage level of this bus signals is V_{DDD}.

23.2.3.3 VDIVIDER ground multiplexer

Because accurate measurement of battery voltage is desired, and the device’s analog ground may be at a different potential than the battery negative terminal, a ground reference multiplexer for the high-voltage divider in the pg_hv is high. This multiplexer can select vssa, rsh, or rsl pins (PACSS_MMIO_PACSS_CTL.HVDIVG_MUX_SEL) for measurement and voltage divider ground (vdiv_ret). If the negative pole of the battery is not connected to vssa or it is desired to avoid potential voltage drops across the vssa pin, rsh or rsl (whichever is connected to the battery negative pole) can be selected.

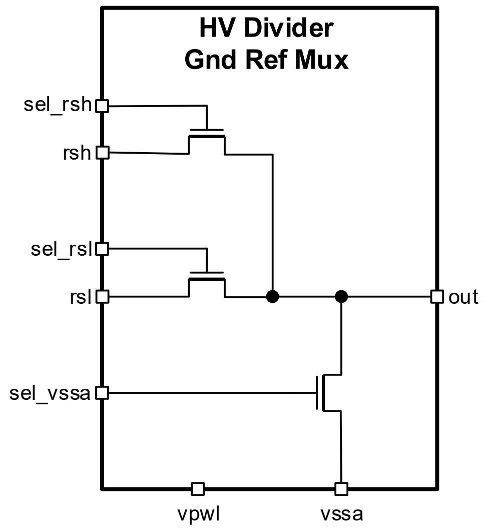


Figure 23-10. VDIVIDER Ground Multiplexer

23.2.3.4 Reset and initialization

VDIVIDER has no registers and latch circuits to hold states. The block ignores low-voltage inputs until the pg_hv input asserts high. When pg_hv is low, the dividers are disabled (leakage current only).

23.2.3.5 Power modes

VDIVIDER is a simple resistor attenuator. Either divider can be disabled to reduce current from the input pins, using the HVSS_RDIV_CTL.RDIV_EN_x registers.

VDIVIDER can also set power domain mode when it is enabled (HVSS_RDIV_CTL.RDIV_ACT_EN register). When RDIV_ACT_EN = 0, the resistor attenuator is enabled in Active and Deep-Sleep modes. When RDIV_ACT_EN = 1, it is enabled in Active mode only.

23.2.3.6 Truth Table

Table 23-4. VDIVIDER (vsense/vdiag) Truth Table

Input = vsense/vdiag			**Output**
pg_hv	**RDIV_EN_x**	**RDIV_SCALE_x**	**vs_div0 / vs_div1**
0	X	X	0
1	0	X	0
1	1	0	INPUT / 16
1	1	1	INPUT / 24

Table 23-5. VDIVIDER Power Domain Mode Truth Table

RDIV_ACT_EN	**Description**
0	Enable in Active and Deep-Sleep modes
1	Enable in Active mode only

23.3 Registers

Table 23-6. List of HVSS Registers

Name	**Offset**	**Qty.**	**Width**	**Description**
HVSS				
HVSS_HVREG_STATUS	0x0000	1	32	HVREG Status Register
HVSS_RDIV_CTL	0x0010	1	32	Resistor Attenuator Control Register
HVSS_LIN_CTL	0x0020	1	32	LIN Phy Control Register
HVSS_LIN_TIMER	0x0024	1	32	LIN Timer Control Register
HVSS_LIN_STATUS	0x0028	1	32	LIN Status Register
HVSS_LIN_INTR	0x0030	1	32	LIN Interrupt Request Register
HVSS_LIN_INTR_SET	0x0034	1	32	LIN Interrupt Set Register
HVSS_LIN_INTR_MASK	0x0038	1	32	LIN Interrupt Mask Register
HVSS_LIN_INTR_MASKED	0x003C	1	32	LIN Interrupt Masked Register
PACSS_MMIO_PACSS_CTL				
HVDIVG_MUX_SEL	0x0000	1	32	High Voltage Divider Ground Reference Mux Select Register

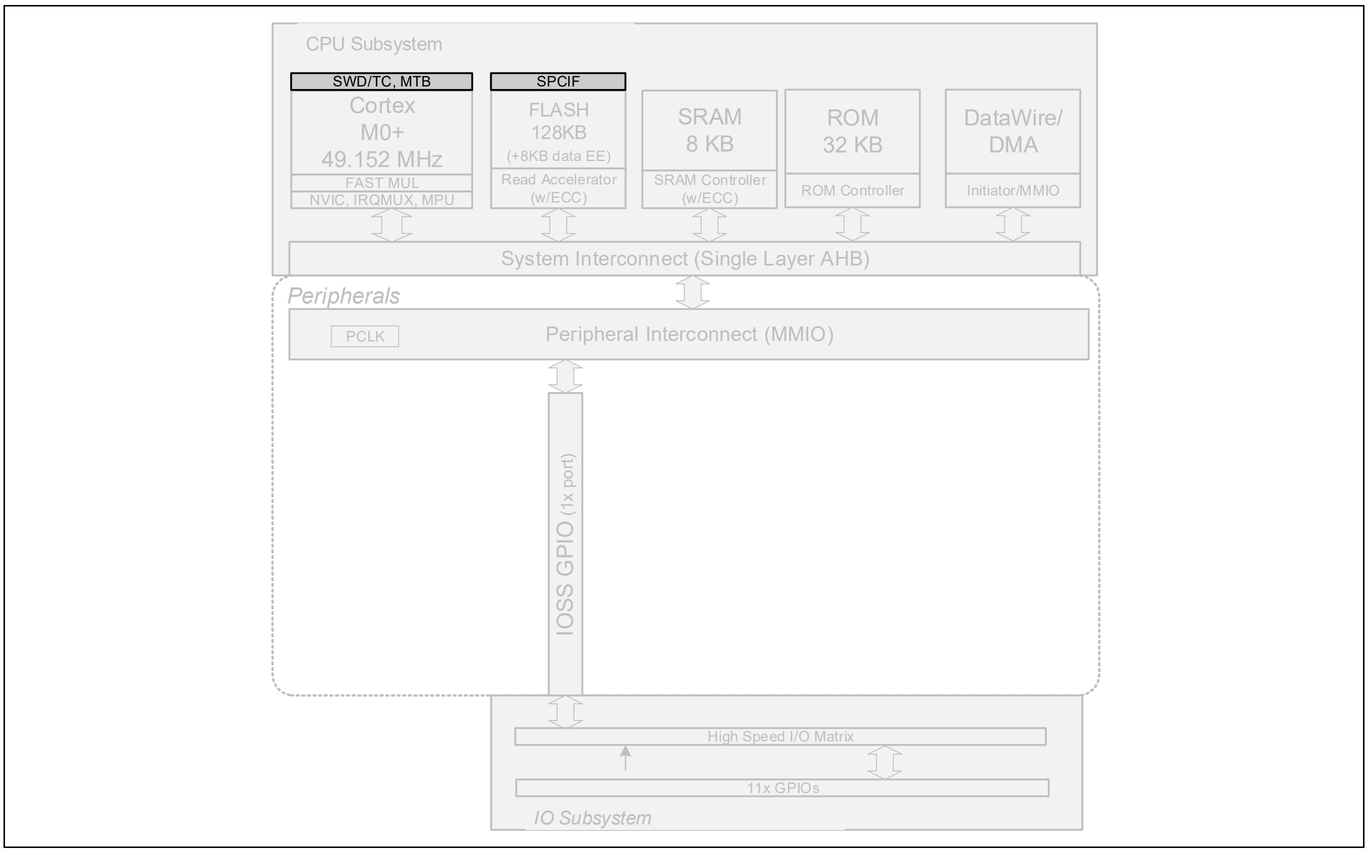
Section G: Program and Debug

This section encompasses the following chapters:

- **"Program and debug interface"** on page 361
- **"Nonvolatile memory programming"** on page 374

Top Level Architecture

Program and Debug Block Diagram



24 Program and debug interface

The PSoC™ 4 HV PA Program and Debug interface provides a communication gateway for an external device to perform programming or debugging. The external device can be a Infineon-supplied programmer and debugger, or a third-party device that supports programming and debugging. The serial wire debug (SWD) interface is used as the communication protocol between the external device and PSoC™ 4 HV PA.

24.1 Features

- Programming and debugging through the SWD interface
- Four hardware breakpoints and two hardware watchpoints while debugging
- Read and write access to all memory and registers in the system while debugging, including the Cortex®-M0+ register bank when the core is running or halted

24.2 Functional description

Figure 24-1 shows the block diagram of the program and debug interface in PSoC™ 4 HV PA. The Cortex®-M0+ debug and access port (DAP) acts as the program and debug interface. The external programmer or debugger, also known as the "host", communicates with the DAP of the PSoC™ 4 HV PA "target" using two pins of the SWD interface - the bidirectional data pin (SWDIO) and the host-driven clock pin (SWDCK). The SWD physical port pins (SWDIO and SWDCK) communicate with the DAP through the high-speed I/O matrix (HSIOM). See the **"I/O system"** on page 141 for details on HSIOM.

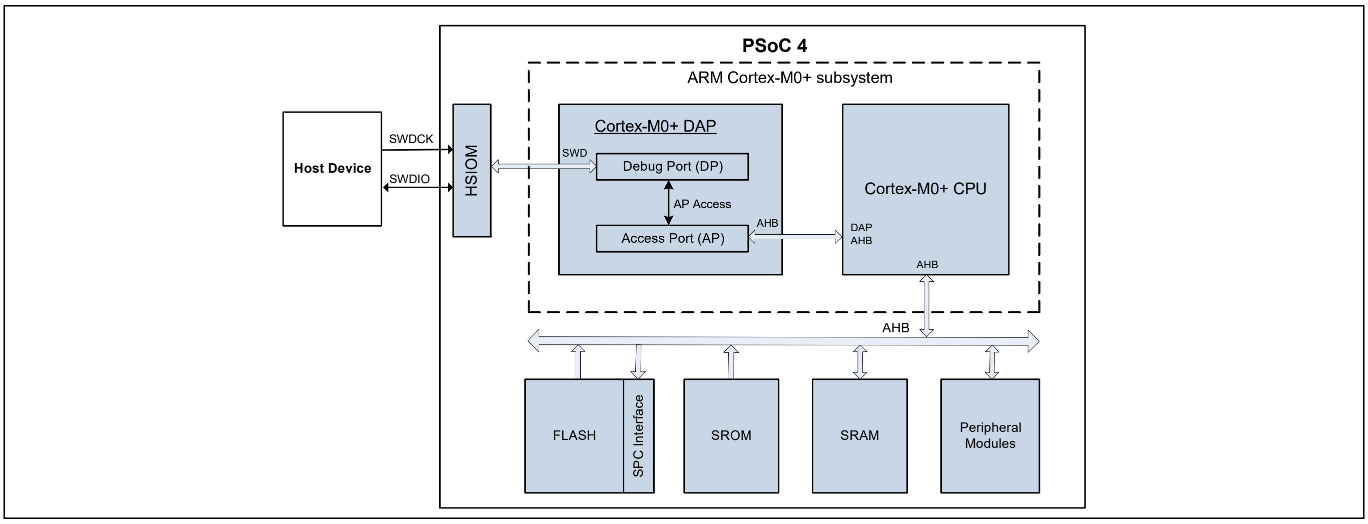
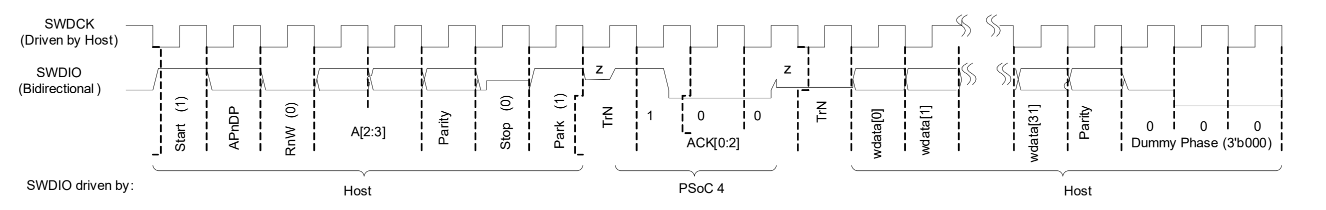


Figure 24-1. Program and debug interface

The DAP communicates with the Cortex-M0+ CPU using the Arm®-specified advanced high-performance bus (AHB) interface. AHB is the systems interconnect protocol used inside the device, which facilitates memory and peripheral register access by the AHB master. The device has two AHB masters – Arm® CM0 CPU core and DAP. The external device can effectively take control of the entire device through the DAP to perform programming and debugging operations.

24.3 Serial wire debug (SWD) interface

PSoC™ 4's Cortex®-M0+ supports programming and debugging through the SWD interface. The SWD interface has two signals: data (SWDIO) and clock for data (SWDCK). The host programmer always drives the clock line, whereas either the programmer or the PSoC™ device drives the data line. The host programmer and PSoC™ communicate in packet format through the SWD interface. Write packet refers to the SWD packet transaction in which the host writes data to PSoC™ 4. Read packet refers to the SWD packet transaction in which the host reads data from PSoC™ 4. The Write packet and Read packet formats are illustrated in **Figure 24-2** and **Figure 24-3**, respectively.



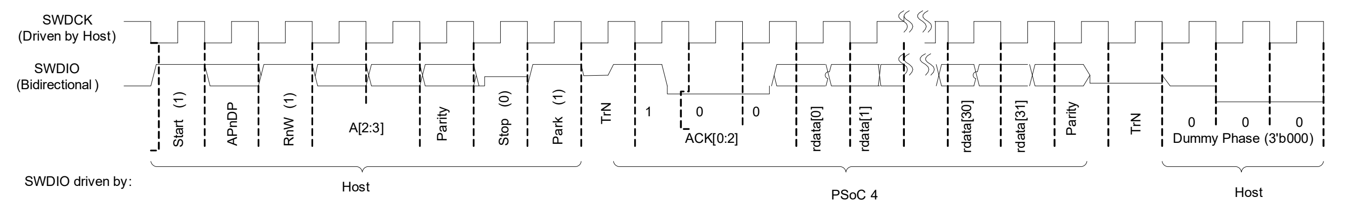
a.) Host Write Operation: Host sends data on the SWDIO line on the falling edge of SWDCK; PSoC 4 reads that data on the next SWDCK rising edge (Example: 8-bit header data, Write data(wdata[31:0]), Dummy phase (3'b000))

b.) Host Read Operation: PSoC 4 sends data on the SWDIO line on the rising edge of SWDCK; host reads that data on the next SWDCK falling edge (Example: ACK data (ACK[2:0])

c.) The host should not drive the SWDIO line during TrN phase. During first TrN phase (½ cycle duration) of SWD packet, PSoC 4 drives the ACK data on the SWDIO line on the rising edge of SWDCK. The host should read the data on the subsequent falling edge of SWDCK. The second TrN phase is 1.5 SWDCK clock cycles. Both PSoC 4 and the host will not drive the line during the entire second TrN phase (indicated as 'z'). Host starts sending the Write data (wdata) on the next falling edge of SWDCK after second TrN phase.

d.) "DUMMY" phase is three SWD clock cycles with SWDIO line low. This DUMMY phase is not part of SWD protocol. The three extra clocks with SWDIO low are required for the Test Controller in PSoC 4 to complete the Read/Write operation when the SWDCK clock is not free-running. For a reliable implementation, include three IDLE clock cycles with SWDIO low for each packet. According to the SWD protocol, the host can generate any number of SWD clock cycles between two packets with SWDIO low.

Figure 24-2. SWD 'Write Packet' timing diagram



a.) Host Write Operation: Host sends data on the SWDIO line on the falling edge of SWDCK; PSoC 4 reads that data on the next SWDCK rising edge (Example: 8-bit header data, Dummy phase (3'b000))

b.) Host Read Operation: PSoC 4 sends data on the SWDIO line on the rising edge of SWDCK; the Host reads that data on the next SWDCK falling edge (Example: ACK data (ACK[2:0], Read data (rdata[31:0]))

c.) The host should not drive the SWDIO line during TrN phase. During first TrN phase (½ cycle duration) of SWD packet, PSoC 4 drives the ACK data on the SWDIO line on the rising edge of SWDCK. The host should read the data on the subsequent falling edge of SWDCK. The second TrN phase is 1.5 SWDCK clock cycles. Both PSoC 4 and the host will not drive the line during the entire second TrN phase (indicated as 'z'). Host starts sending the Dummy phase (3'b000) on the next falling edge of SWDCK after the second TrN phase.

d.) "DUMMY" phase is three SWD clock cycles with SWDIO line low. This DUMMY phase is not part of SWD protocol. The three extra clocks with SWDIO low are required for the Test Controller in PSoC 4 to complete the Read/Write operation when the SWDCK clock is not free-running. For a reliable implementation, include three IDLE clock cycles with SWDIO low for each packet. According to the SWD protocol, the host can generate any number of SWD clock cycles between two packets with SWDIO low.

Figure 24-3. SWD 'Read Packet' timing diagram

A complete data transfer requires 46 clocks (not including the optional three dummy clock cycles in **Figure 24-2** and **Figure 24-3**). Each data transfer consists of three phases:

- **Packet request** – External host programmer issues a request to PSoC™ 4.
- **Acknowledge response** – PSoC™ 4 sends an acknowledgement to the host.
- **Data** – This is valid only when a packet request is followed by a valid (OK) acknowledge response.

The data transfer is either:

- PSoC™ 4 to host, following a read request – RDATA
- Host to PSoC™ 4, following a write request – WDATA

In **Figure 24-2** and **Figure 24-3**, the following sequence occurs:

1. The start bit initiates a transfer; it is always logic ‘1’.
2. The APnDP bit determines whether the transfer is an AP access, ‘1’, or a DP access, ‘0’.
3. The next bit is RnW, which is ‘1’ for a read from PSoC™ 4, or ‘0’ for a write to PSoC™ 4.
4. The ADDR bits (A[3:2]) are register select bits for access port or debug port. See **Table 24-2** and **Table 24-3** for address bit definitions.
5. The parity bit has the parity of APnDP, RnW, and ADDR. This is even parity bit. If the number of logical 1s in these bits is odd, then parity must be ‘1’, otherwise, it is ‘0’.

If the parity bit is not correct, the header is ignored by the target device; there is no ACK response. For host implementation, the programming operation should be stopped and tried again by doing a device reset.

1. The stop bit is always logic ‘0’.
2. The park bit is always logic’1’ and should be driven high by the host.
3. The ACK bits are the device-to-host response.

Possible values are shown in **Table 24-1**. Note that the ACK in the current SWD transfer reflects the status of the previous transfer. OK ACK means the previous packet was successful. WAIT response indicates that the previous packet transaction is not yet complete. For a FAULT operation, the programming operation should be aborted immediately.

Table 24-1. SWD Transfer ACK Response Decoding

Response	ACK[2:0]
OK	3b001
WAIT	3b010
FAULT	3b100

   a) For a WAIT response, if the transaction is a read, the host ignores the data read in the data phase. PSoC™ 4 does not drive the line and the host must not check the parity bit as well.
   b) For a WAIT response, if the transaction is a write, PSoC™ 4 ignores the data phase. However, the host must still send the data to be written from an implementation standpoint. The parity data corresponding to the data should also be sent by the host.
   c) A WAIT response indicates that the PSoC™ device is processing the previous transaction. The host can try for a maximum of four continuous WAIT responses to see if an OK response is received, failing which, it can abort the programming operation and retry.
   d) For a FAULT response, the programming operation should be aborted and retried by doing a device reset.
4. The data phase includes a parity bit (even parity, similar to the packet request phase).
   a) For a read data packet, if the host detects a parity error, then it must abort the programming operation and restart.
   b) For a write data packet, if the PSoC™ 4 detects a parity error in the data packet sent by the host, it generates a FAULT ACK response in the next packet.

5. Turnaround (TrN) phase: According to the SWD protocol, the TrN phase is used both by the host and PSoC™ 4 to change the drive modes on their respective SWDIO line. During the first TrN phase after packet request, PSoC™ 4 drives the ACK data on the SWDIO line on the rising edge of SWDCK in the TrN phase. This ensures that the host can read the ACK data on the next falling edge. Thus, the first TrN cycle is only for half-cycle duration. The second TrN phase is one-and-a-half cycle long. Neither the host nor PSoC™ 4 should drive SWDIO line during both phases as indicated by 'z' in **Figure 24-2** and **Figure 24-3**.
6. The address, ACK, and read and write data are always transmitted least significant bit (LSB) first.
7. At the end of each SWD packet in **Figure 24-2** and **Figure 24-3**, there is a "DUMMY" phase, which is three SWD clock cycles with SWDIO line held low. The dummy phase is not part of the SWD protocol. The three extra clocks with SWDIO low are required for the test controller in PSoC™ 4 to complete the read/write operation when the SWDCK clock is not free-running. For reliable implementation, include three IDLE clock cycles with SWDIO low for each packet. According to the SWD protocol, the host can generate any number of SWD clock cycles between two packets when SWDIO is low.

Note: *The SWD interface can be reset anytime during programming, by clocking 50 or more cycles with SWDIO high. To return to the idle state, SWDIO must be clocked low once. The host programmer can begin a new SWD packet transaction from the idle state.*

24.4 Cortex®-M0+ debug and access port (DAP)

The Cortex®-M0+ program and debug interface includes a Debug Port (DP) and an Access Port (AP), which combine to form the DAP. The debug port implements the state machine for the SWD interface protocol that enables communication with the host device. It also includes registers for the configuration of access port, DAP identification code, and so on. The access port contains registers that enable the external device to access the Cortex®-M0+ DAP-AHB interface. Typically, the DP registers are used for a one-time configuration or for error detection purposes, and the AP registers are used to for programming and debugging operations. Complete architecture details of the DAP is available in the *Arm® Debug Interface v5 Architecture Specification*.

24.4.1 Debug Port (DP) registers

Table 24-2 shows the Cortex®-M0+ DP registers used for programming and debugging, along with the corresponding SWD address bit selections. The APnDP bit is always zero for DP register accesses. Two address bits (A[3:2]) are used for selecting among the different DP registers. Note that for the same address bits, different DP registers can be accessed depending on whether it is a read or a write operation. See the *Arm® Debug Interface v5 Architecture Specification* for details on all of the DP registers.

Table 24-2. Main Debug Port (DP) Registers

Register	**APnDP**	**Address A[3:2]**	**RnW**	**Full name**	**Register functionality**
ABORT	0 (DP)	2b00	0 (W)	AP Abort Register	This register is used to force a DAP abort and to clear the error and sticky flag conditions.
IDCODE	0 (DP)	2b00	1 (R)	Identification Code Register	This register holds the SWD ID of the Cortex®-M0+ CPU, which is 0x0BB11477.
CTRL/STAT	0 (DP)	2b01	X (R/W)	Control and Status Register	This register allows control of the DP and contains status information about the DP.
SELECT	0 (DP)	2b10	0 (W)	AP Select Register	This register is used to select the current AP. In PSoC™ 4, there is only one AP, which interfaces with the DAP AHB.
RDBUFF	0 (DP)	2b11	1 (R)	Read Buffer Register	This register holds the result of the last AP read operation.

24.4.2 Access Port (AP) registers

Table 24-3 lists the main Cortex®-M0+ AP registers that are used for programming and debugging, along with the corresponding SWD address bit selections. The APnDP bit is always ‘1’ for AP register accesses. Two address bits (A[3:2]) are used for selecting the different AP registers.

Table 24-3. Main Access Port (AP) Registers

Register	APnDP	Address A[3:2]	RnW	Full name	Register functionality
CSW	1 (AP)	2b00	X (R/W)	Control and Status Word Register (CSW)	This register configures and controls accesses through the memory access port to a connected memory system (which is the PSoC™ 4 Memory map).
TAR	1 (AP)	2b01	X (R/W)	Transfer Address Register	This register is used to specify the 32-bit memory address to be read from or written to.
DRW	1 (AP)	2b11	X (R/W)	Data Read and Write Register	This register holds the 32-bit data read from or to be written to the address specified in the TAR register.

24.5 Programming the PSoC™ 4 device

PSoC™ 4 is programmed using the following sequence. Refer to the *PSoC™ 4 HV PA Programming Specifications* for complete details on the programming algorithm, timing specifications, and hardware configuration required for programming.

1. Acquire the SWD port in PSoC™ 4 HV PA.
2. Enter the programming mode.
3. Execute the device programming routines such as Silicon ID Check, Flash Programming, Flash Verification, and Checksum Verification.

24.5.1 SWD port acquisition

24.5.1.1 SWD port acquire sequence

The first step in device programming is for the host to acquire the target's SWD port. The host first performs a device reset by asserting the external reset (XRES) pin. After removing the XRES signal, the host must send an SWD connect sequence for the device within the acquire window to connect to the SWD interface in the DAP. The pseudo code for the sequence is given here.

Code 1. SWD Port Acquire Pseudo Code

```
ToggleXRES(); // Toggle XRES pin to reset device

//Execute Arm's connection sequence to acquire SWD-port
do
{
        SWD_LineReset(); //perform a line reset (50+ SWDCK clocks with SWDIO high)
        ack = Read_DAP ( IDCODE, out ID); //Read the IDCODE DP register

}while ((ack != OK) && time_elapsed < 1.5 ms); //retry connection until OK ACK or timeout

if (time_elapsed >= 1.5 ms) return FAIL; //check for acquire time out

if (ID != CM0P_ID) return FAIL; //confirm SWD ID of Cortex-M0+ CPU. (0x0BC11477)
```

In this pseudo code, SWD_LineReset() is the standard Arm® command to reset the debug access port. It consists of more than 49 SWDCK clock cycles with SWDIO high. The transaction must be completed by sending at least one SWDCK clock cycle with SWDIO asserted LOW. This sequence synchronizes the programmer and the chip. Read_DAP() refers to the read of the IDCODE register in the debug port. The sequence of line reset and IDCODE read should be repeated until an OK ACK is received for the IDCODE read or a timeout (1.5 ms) occurs. The SWD port is said to be in the acquired state if an OK ACK is received within the time window and the IDCODE read matches with that of the Cortex®-M0+ DAP.

24.5.2 SWD programming mode entry

After the SWD port is acquired, the host must enter the device programming mode within a specific time window. This is done by setting the TEST_MODE bit (bit 31) in the test mode control register (MODE register). The debug port should also be configured before entering the device programming mode. Timing specifications and pseudo code for entering the programming mode are detailed in the *PSoC™ 4 HV PA Programming Specifications* document. The minimum required clock frequency for the Port Acquire step and Programming Mode Entry step to succeed is 1.5 MHz.

24.5.3 SWD programming routines executions

When the device is in programming mode, the external programmer can start sending the SWD packet sequence for performing programming operations such as flash erase, flash program, checksum verification, and so on. The programming routines are explained in the **"Nonvolatile memory programming"** on page 374. The exact sequence of calling the programming routines is given in the *PSoC™ 4 HV PA Programming Specifications*.

24.6 PSoC™ 4 SWD debug interface

Cortex®-M0+ DAP debugging features are classified into two types: invasive debugging and noninvasive debugging. Invasive debugging includes program halting and stepping, breakpoints, and data watchpoints. Noninvasive debugging includes instruction address profiling and device memory access, which includes the flash memory, SRAM, and other peripheral registers.

The DAP has three major debug subsystems:

- Debug Control and Configuration registers
- Breakpoint Unit (BPU) – provides breakpoint support
- Debug Watchpoint (DWT) – provides watchpoint support. Cortex®-M0+ Debug does not support trace

See the *Armv6-M Architecture Reference Manual* for complete details on the debug architecture.

24.6.1 Debug control and configuration registers

The debug control and configuration registers are used to execute firmware debugging. The registers and their key functions are as follows. See the *Armv6-M Architecture Reference Manual* for complete bit level definitions of these registers.

- Debug Halting Control and Status Register (CM0P_DHCSR) – This register contains the control bits to enable debug, halt the CPU, and perform a single-step operation. It also includes status bits for the debug state of the processor.
- Debug Fault Status Register (CM0P_DFSR) – This register describes the reason a debug event has occurred and includes debug events, which are caused by a CPU halt, breakpoint event, or watchpoint event.
- Debug Core Register Selector Register (CM0P_DCRSR) – This register is used to select the general-purpose register in the Cortex®-M0+ CPU to which a read or write operation must be performed by the external debugger.
- Debug Core Register Data Register (CM0P_DCRDR) – This register is used to store the data to write to or read from the register selected in the CM0P_DCRSR register.
- Debug Exception and Monitor Control Register (CM0P_DEMCR) – This register contains the enable bits for global debug watchpoint (DWT) block enable, reset vector catch, and hard fault exception catch.

24.6.2 Breakpoint unit (BPU)

The BPU provides breakpoint functionality on instruction fetches. The Cortex®-M0+ DAP in PSoC™ 4 HV PA supports up to four hardware breakpoints. Along with the hardware breakpoints, any number of software breakpoints can be created by using the BKPT instruction in the Cortex®-M0+. The BPU has two types of registers.

- The breakpoint control register (CM0P_BP_CTRL) is used to enable the BPU and store the number of hardware breakpoints supported by the debug system (four for CM0 DAP in the PSoC™ 4 HV PA).
- Each hardware breakpoint has a Breakpoint Compare Register (CM0P_BP_COMPx). It contains the enable bit for the breakpoint, the compare address value, and the match condition that will trigger a breakpoint debug event. In a typical use case, when an instruction fetch address matches the compare address of a breakpoint, a breakpoint event is generated and the processor is halted.

24.6.3 Data watchpoint (DWT)

The DWT provides watchpoint support on a data address access or a program counter (PC) instruction address. The DWT supports two watchpoints. It also provides external program counter sampling using a PC sample register, which can be used for noninvasive coarse profiling of the program counter. The most important registers in the DWT are as follows.

- The watchpoint compare (CM0P_DWT_COMPx) registers store the compare values that are used by the watchpoint comparator for the generation of watchpoint events. Each watchpoint has an associated DWT_COMPx register.
- The watchpoint mask (CM0P_DWT_MASKx) registers store the ignore masks applied to the address range matching in the associated watchpoints.
- The watchpoint function (CM0P_DWT_FUNCTIONx) registers store the conditions that trigger the watchpoint events. They may be program counter watchpoint event or data address read/write access watchpoint events. A status bit is also set when the associated watchpoint event occurs.
- The watchpoint comparator PC sample register (CM0P_DWT_PCSR) stores the current value of the program counter. This register is used for coarse, noninvasive profiling of the program counter register.

24.6.4 Debugging the PSoC™ 4 device

The host debugs the target PSoC™ 4 HV PA by accessing the debug control and configuration registers, registers in the BPU, and registers in the DWT. All registers are accessed through the SWD interface; the SWD debug port (SW-DP) in the Cortex®-M0+ DAP converts the SWD packets to appropriate register access through the DAP-AHB interface.

The first step in debugging the target PSoC™ 4 is to acquire the SWD port. The acquire sequence consists of an SWD line reset sequence and read of the DAP SWDID through the SWD interface. The SWD port is acquired when the correct CM0 DAP SWDID is read from the target device. For the debug transactions to occur on the SWD interface, the corresponding pins should not be used for any other purpose. See the **"I/O system"** on page 141 to understand how to configure the SWD port pins, allowing them to be used only for SWD interface or for other functions such as LCD and GPIO. If debugging is required, the SWD port pins should not be used for other purposes. If only programming support is needed, the SWD pins can be used for other purposes.

When the SWD port is acquired, the external debugger sets the C_DEBUGEN bit in the DHCSR register to enable debugging. Then, the different debugging operations such as stepping, halting, breakpoint configuration, and watchpoint configuration are carried out by writing to the appropriate registers in the debug system.

Debugging the target device is also affected by the overall device protection setting, which is explained in the **"Device security and register protection"** on page 137. Only the OPEN protected mode supports device debugging. The external debugger and the target device connection is not lost for a device transition from Active mode to either Sleep or Deep Sleep modes. When the device enters the Active mode from either Deep Sleep or Sleep modes, the debugger can resume its actions without initiating a connect sequence again.

24.7 Accessing PSoC™ memory and registers

The DCRDR and DCRSR are used to access the PSoC™ memory and registers. The register and memory access are 32 bits wide.

To use the registers to read the contents of a register, perform the following steps:

1. Set the C_DEBUGEN and C_HALT bits of the DHCSR. This enables the debug and halts the core.
2. Wait for the S_HALT bit of the DHCSR to be set. This indicates that the core is halted.
3. Write to the DCRSR, with the REGSEL value indicating the required register and the REGWnR bit as '0' indicating a read access.
4. This write clears the DHCSR.S_REGRDY bit to 0.
5. Poll DHCSR until DHCSR.S_REGRDY reads as one. This shows that the processor has transferred the value of the selected register to DCRDR.
6. Read the required value from DCRDR.
7. To write to a register, perform the following steps:
8. Make sure the processor is halted by following steps 1 and 2 mentioned above.
9. Write the required word to DCRDR.
10. Write to the DCRSR, with the REGSEL value indicating the required register, and the REGWnR bit as '1' indicating a write access.

    This write clears the DHCSR S_REGRDY bit to 0.
11. Poll DHCSR until DHCSR.S_REGRDY reads as one. This shows that the processor has transferred the DCRDR value to the selected register.

The Memory Access Port (MEM-AP) provides access to the memory through the DAP. All accesses to a MEM-AP are made through the MEM-AP registers. All registers are 32 bits wide. The important registers required for memory access include:

- **Control/Status Word Register (CSW)** – The CSW configures and controls accesses through the MEM-AP to or from a connected memory system.
- **Transfer Address Register (TAR)** – The TAR holds the memory address to be accessed.
- **Data Read/Write Register (DRW)** – The DRW holds a 32-bit data value. In write mode, the DRW holds the value to write for the current transfer to the address specified in TAR[31:0]. In read mode, the DRW holds the value read in the current transfer from the address specified in TAR[31:0].
- **Configuration Register (CFG)** – The CFG provides information about the configuration of the MEM-AP implementation. It indicates whether memory accesses by the MEM-AP are big-endian or little-endian.
- **Debug Base Address Register (BASE)** – The BASE provides an index into the connected memory-mapped resource. This index value points to one of the following: the start of a set of debug registers or a ROM table that describes the connected debug components.

For more details on the Memory Access Port and registers, see the *Arm® Debug Interface v5 Architecture Specification*.

24.8 Multi-drop serial wire debug port (SW-DP)

The Arm® Debug Interface Architecture Specification: ADIv5.1 introduces the serial wire debug (SWD) protocol version 2 that extends the serial wire debug protocol, adding multi-drop capability. The following is a description of the multi-drop architecture:

- Enables a two-wire host connection to communicate simultaneously with multiple devices.
- Permits an effectively unlimited number of devices to be connected simultaneously, subject to electrical constraints.
- Is largely backwards-compatible, because provision for multi-drop support in a device does not break point-to-point compatibility with existing host equipment that does not support the multi-drop extensions.
- Permits a device to power down completely, during the time that the device is not selected.
- Prevents multiple devices from driving the wire simultaneously, and continues to support the wire being actively driven both HIGH and LOW, maintaining a high maximum clock speed.
- Permits multi-drop connections incorporating devices that do not implement the SWD protocol.

Figure 24-4 shows an example of multi-drop physical connection.

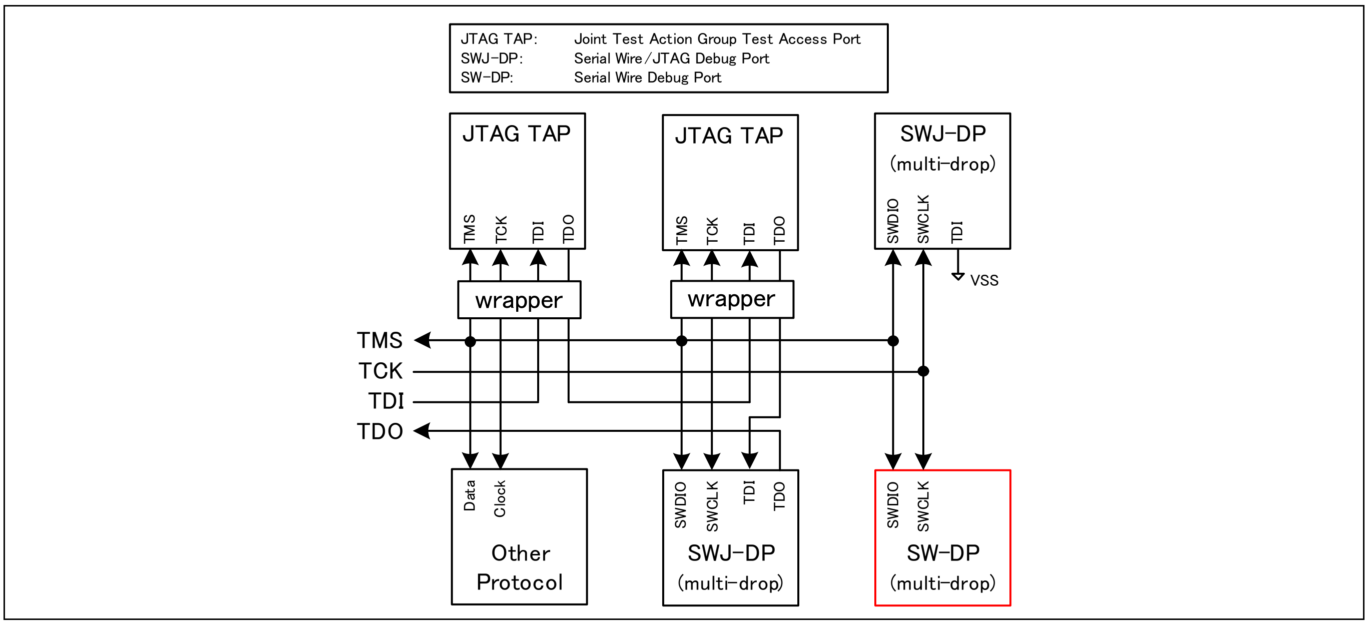


Figure 24-4. Multi-Drop Physical Connection Example

24.8.1 How to support

To support multi-drop SW-DP, a debugger must first place the target into dormant state, and then transition to the required operating state.

Using dormant state allows the target to be placed into a quiescent mode, allowing devices to inter-operate with other devices implementing other protocols. For details on placing the target into the dormant state, see the *Arm® Debug Interface v5 Architecture Specification*.

24.8.2 Limitations of multi-drop

24.8.2.1 System configuration

Each device must be configured with a unique target ID, which includes a 4-bit instance ID, to differentiate between otherwise identical targets. This places a limit of 16 such targets in any system, and indicates that identical devices must be configured before they are connected together to ensure that their instance IDs do not conflict.

24.8.2.2 Auto-detection of target

It is not possible to interrogate a multi-drop SWD system that includes multiple devices to establish which devices are connected. Because all devices are selected when coming out of a line reset, no communication with a device is possible without prior selection of that target using its target ID. Therefore, connection to a multi-drop SWD system that includes multiple devices requires either of the following:

- The host has prior knowledge of the devices in the system and is configured before target connection.
- The host attempts auto-detection by issuing a target select command for each of the devices it is configured to support. While this is likely to involve a large number of target select commands, it must be possible to iterate through all the supported devices in a reasonable time for the debug tool users.

Note: This means that debug tools cannot connect seamlessly to targets in a multi-drop SWD system that they have never seen before. However, if the debug tools can be provided with the target ID of such targets by the user, then the contents of the target can be auto-detected as normal.

24.8.2.3 Compatibility with SWD Protocol Version 1

To protect against multiple selected devices all driving the line simultaneously, SWD protocol version 2 requires the DP to be in dormant state out of power-up reset; this means the start-up state differs from that in the SWD protocol version 1. If SWD operation is then selected, operation becomes compatible with SWD protocol version 1.

24.9 Registers

Table 24-4. List of Registers

Register name	Description
CM0P_DHCSR	Debug Halting Control and Status Register
CM0P_DFSR	Debug Fault Status Register
CM0P_DCRSR	Debug Core Register Selector Register
CM0P_DCRDR	Debug Core Register Data Register
CM0P_DEMCR	Debug Exception and Monitor Control Register
CM0P_BP_CTRL	Breakpoint Control Register
CM0P_BP_COMPx	Breakpoint Compare Register
CM0P_DWT_COMPx	Watchpoint Compare Register
CM0P_DWT_MASKx	Watchpoint Mask Register
CM0P_DWT_FUNCTIONx	Watchpoint Function Register
CM0P_DWT_PCSR	Watchpoint Comparator PC Sample Register

25 Nonvolatile memory programming

Nonvolatile memory programming refers to the programming of flash memory in the PSoC™ 4 HV PA device. This chapter explains the different functions that are part of device programming, such as erase, write, program, and checksum calculation. Infineon-supplied programmers and other third-party programmers can use these functions to program the PSoC™ 4 HV PA device with the data in an application hex file. They can also be used to perform bootload operations where the CPU will update a portion of the flash memory.

25.1 Features

- Supports programming through the debug and access port (DAP) and Cortex®-M0+ CPU
- Supports both blocking and non-blocking flash program and erase operations from the Cortex®-M0+ CPU

25.2 Functional description

Flash programming operations are implemented as system calls. System calls are executed out of SROM in the privileged mode of operation. The user has no access to read or modify the SROM code. The DAP or the CM0+ CPU requests the system call by writing the function opcode and parameters to the System Performance Controller Interface (SPCIF) input registers, and then requesting the SROM to execute the function. Based on the function opcode, the System Performance Controller (SPC) executes the corresponding system call from SROM and updates the SPCIF status register. The DAP or the CPU should read this status register for the pass/fail result of the function execution. As part of function execution, the code in SROM interacts with the SPCIF to do the actual flash programming operations (See the **"Flash Memory"** on page 67 for more information of the SPCIF).

PSoC™ 4 HV PA flash is programmed using a Program Erase Program (PEP) sequence that happens automatically in the SROM routine. The flash cells are all programmed to a known state, erased, and then the selected bits are programmed. This sequence increases the life of the flash by balancing the stored charge. When writing to flash the data is first copied to a page latch buffer. The flash write functions are then used to transfer this data to flash memory.

External programmers program the flash memory in PSoC™ 4 HV PA using the SWD protocol by sending the commands to the DAP. The programming sequence for the PSoC™ 4 HV PA device with an external programmer is given in the PSoC™ 4 HV PA Programming Specifications. Flash memory can also be programmed by the CM0+ CPU by accessing the relevant registers through the AHB interface. This type of programming is typically used to update a portion of the flash memory as part of a bootload operation or other application requirements, such as updating a lookup table stored in flash memory, or data logging. All write operations to flash memory, whether from the DAP or from the CPU, are done through the SPCIF.

Note: It can take as much as 20 milliseconds to write to flash. During this time the device should not be reset, or unexpected changes may be made to portions of the flash. Reset sources include XRES pin, software reset, and watchdog (see the **"Reset system and interrupts"** on page 133); make sure these are not inadvertently activated.

25.3 System Call implementation

A system call consists of the following items:

- Opcode: A unique 8-bit opcode
- Parameters: Two 8-bit parameters are mandatory for all system calls. These parameters are referred to as key1 and key2, and are defined as follows:
  - key1 = 0xB6
  - key2 = 0xD3 + Opcode
  - The two keys are passed to ensure that the user system call is not initiated by mistake. If the key1 and key2 parameters are not correct, the SROM does not execute the function, and returns an error code. Apart from these two parameters, additional parameters may be required depending on the specific function being called.
- Return values: Some system calls also return a value on completion of their execution, such as the silicon ID or a checksum.
- Completion status: Each system call returns a 32-bit status that the CPU or DAP can read to verify success or determine the reason for failure.

25.4 Blocking and Non-Blocking System Calls

System call functions can be categorized as blocking or non-blocking based on the nature of their execution. Blocking system calls are those where the CPU cannot execute any other task in parallel other than the execution of the system call. When a blocking system call is called from a process, the CPU jumps to the corresponding code in SROM. When the execution is complete, the original thread execution resumes. Non-blocking system calls allow the CPU to execute some other code in parallel and communicate the completion of interim system call tasks to the CPU through an interrupt.

Non-blocking system calls are only used when the CPU initiates the system call. The DAP will only use blocking system calls during the programming mode and the CPU is halted during this process.

The three non-blocking system calls are Non-Blocking Write Row, Non-Blocking Program Row, and Resume Non-Blocking, respectively. All other system calls are blocking.

The CPU cannot execute code from code flash while doing an erase or program operation. Doing so results in an undefined behavior, which may return a bus error and/or trigger a hard fault when the flash fetch operation is being done. For this reason, blocking system calls must be used when programming code flash.

Data flash is not expected to contain executable code, and can safely be updated using non-blocking system calls.

For more information of code flash and data flash, see the **"Flash Memory"** on page 67.

The SPC block generates the properly sequenced high-voltage pulses required for erase and program operations of the flash memory. When a non-blocking function is called, the SPC timer triggers its interrupt when each of the sub-operations in a write or program operation is complete. Call the Resume Non-Blocking function from the SPC interrupt service routine (ISR) to ensure that the subsequent steps in the system call are completed. The SPC interrupt is triggered once in the case of a non-blocking program function or thrice in a non-blocking write operation. Similarly, the Resume Non-Blocking function call in the SPC ISR should be called once in a non-blocking program operation and thrice in a non-blocking write operation.

25.4.1 Performing a system call

The steps to initiate a system call are as follows:

1. Set up the function parameters: The two possible methods for preparing the function parameters (key1, key2, or additional parameters) are:
   a) Write the function parameters to the CPUSS_SYSARG register: This method is used for functions that retrieve their parameters from the CPUSS_SYSARG register. The 32-bit CPUSS_SYSARG register must be written with the parameters in the sequence specified in the respective system call table.
   b) Write the function parameters to SRAM: This method is used for functions that retrieve their parameters from SRAM. The parameters should first be written in the specified sequence to consecutive SRAM locations. Then, the starting address of the SRAM, which is the address of the first parameter, should be written to the CPUSS_SYSARG register. This starting address should always be a word-aligned (32-bit) address. The system call uses this address to fetch the parameters.
2. Specify the system call using its opcode and initiating the system call: The 8-bit opcode should be written to the SYSCALL_COMMAND bits ([15:0]) in the CPUSS_SYSREQ register. The opcode is placed in the lower eight bits [7:0] and 0x00 be written to the upper eight bits [15:8]. To initiate the system call, set the SYSCALL_REQ bit (31) in the CPUSS_SYSREG register. Setting this bit triggers a non-maskable interrupt that jumps the CPU to the SROM code referenced by the opcode parameter.
3. Wait for the system call to finish executing: When the system call begins execution, it sets the PRIVILEGED bit in the CPUSS_SYSREQ register. This bit can be set only by the system call, not by the CPU or DAP. The DAP should poll the PRIVILEGED and SYSCALL_REQ bits in the CPUSS_SYSREG register continuously to check whether the system call is completed. Both these bits are cleared on completion of the system call. The maximum execution time is one second. If these two bits are not cleared after one second, the operation should be considered a failure and aborted without executing the remaining steps. Note that unlike the DAP, the CPU application code cannot poll these bits during system call execution. This is because the CPU executes code out of the SROM during the system call. The application code can check only the final function pass/fail status after the execution returns from SROM.
4. Check the completion status: After the PRIVILEGED and SYSCALL_REQ bits are cleared to indicate completion of the system call, the CPUSS_SYSARG register should be read to check for the status of the system call. If the 32-bit value read from the CPUSS_SYSARG register is 0xAXXXXXXX (where ‘X’ denotes don’t care hex values), the system call was successfully executed. For a failed system call, the status code is 0xF00000YY where YY indicates the reason for failure. See **Table 25-1** for the complete list of status codes and their description.
5. Retrieve the return values: For system calls that return values such as silicon ID and checksum, the CPU or DAP should read the CPUSS_SYSREG and CPUSS_SYSARG registers to fetch the values returned.

25.5 System calls

Table 25-1 lists all the system calls supported in PSoC™ 4 HV PA along with the function description and availability in device protection modes. See the **"Device security and register protection"** on page 137 for more information on the device protection settings. Note that some system calls cannot be called by the CPU as given in the table. Detailed information on each of the system calls follows the table.

Table 25-1. List of System Calls

System Call	Op Code	Description	DAP Access			CPU Access
Silicon ID	0x00	Returns the device Silicon ID, Family ID, and Revision ID	✓	✓	–	✓
Load Flash Bytes	0x04	Loads data to the page latch buffer to be programmed later into the flash row, in 8 byte granularity, for a row size of 128 bytes	✓	–	–	✓
Write Row	0x05	Erases and then programs a row of flash with data in the page latch buffer	✓	–	–	✓
Program Row	0x06	Programs a row of flash with data in the page latch buffer	✓	–	–	✓
Non-Blocking Write Row	0x07	Erases and then programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access	–	–	–	✓
Non-Blocking Program Row	0x08	Programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access	–	–	–	✓
Resume Non-Blocking	0x09	Resumes a non-blocking write row or non-blocking program row. This function is meant only for CPU access	–	–	–	✓
Erase All	0x0A	Erases all user code in the flash main and work arrays, chip protection, and all Flash Row Level Write Protection data in SFlash.	✓	–	–	–
Checksum	0x0B	Calculates the checksum over the entire flash memory (user and supervisory area) or checksums a single row of flash.	✓	✓	–	✓
Write Protection	0x0D	This programs both the flash row-level protection settings and the device protection settings in the supervisory flash row.	✓	✓	–	–
Configure Clock	0x15	Sets the clock to required frequency for flash program/erase operations.	✓	✓	–	✓
Write User SFlash Row	0x18	Writes a user row of supervisory flash.	✓	–	–	✓

Table 25-1. List of System Calls (continued)

System Call	Op Code	Description	DAP Access			CPU Access
Soft Reset	0x1B	Issues software reset by setting the CM0_AIRCR.SYSRESETREQ bit.	✓	✓	–	–
Bulk Erase	0x1D	Erases single or all macro of regular flash of the specific flash controller in bulk mode.	✓	–	–	✓
Specify External Clock Frequency	0x1E	Specifies frequency on EXTCLK/ECO1) input to allow program/erase operations while being clocked from this clock source.	✓	✓	–	✓

1) ECO is not available as a clock for the PSoC™ 4 HV PA device. It will be available in future products.

25.5.1 Silicon ID

This function returns a 12-bit family ID, 16-bit silicon ID, an 8-bit revision ID, and the current device protection mode. These values are returned to the CPUSS_SYSARG and CPUSS_SYSREQ registers. Parameters are passed through the CPUSS_SYSARG and CPUSS_SYSREQ registers.

Parameters

Table 25-2. Parameters

Address	Value to be Written	Description
CPUSS_SYSARG Register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD3	Key2
Bits [31:16]	0x0000	Not used
CPUSS_SYSREQ register		
Bits [15:0]	0x0000	Silicon ID opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Table 25-3. Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [7:0]	Silicon ID Lo	See the PSoC™ 4 HV PA datasheet for Silicon ID values for different part numbers.
Bits [15:8]	Silicon ID Hi	
Bits [19:16]	Minor Revision ID	See the PSoC™ 4 HV PA Programming Specifications for these values.
Bits [23:20]	Major Revision ID	
Bits [27:24]	0xXX	Not used (don't care)
Bits [31:28]	0xA	Success status code
Bits [31:0]	0xF00000YY (Y = failure code)	Failure status code (See **"System Call Status"** on page 399)
CPUSS_SYSREQ register		
Bits [11:0]	Family ID	Family ID is 0xC2 for PSoC™ 4 HV PA
Bits [15:12]	Chip Protection	See the **"Device security and register protection"** on page 137.
Bits [31:16]	0xXXXX	Not used

25.5.2 Load Flash Bytes

This API loads the page latch buffer with data to be programmed into a row of flash. Data is programmed into the page latch buffer starting at the location specified by the Byte Addr input parameter. The Byte Addr parameter together with the Load Size parameter must not exceed the size of the page latch. To write the entire page latch buffer, set the Byte Addr to zero and Load Size to N-1 where N is the size of the page latch.

Flash write width is 64 bits. As a result, Load Size and Byte Addr have 8-byte granularity.

Data programmed into the page latch buffer will remain until a program is performed, which will clear the page latch contents.

Parameters

Table 25-4. Parameters

Address	Value to be Written	Description
SRAM Address - 32’hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD7	Key2
Bits [23:16]	Byte Addr	Start address of page latch buffer to write data Supported values: 0x00 - Byte 0 of latch buffer …. 0x78 - Byte 120 of latch buffer
Bits [30:24]	Flash Macro Select	0x00 - Flash Macro 0 0x01 - Flash Macro 1 0x02 - Flash Macro 2 (Not used in PSoC™ 4 HV PA) 0x03 - Flash Macro 3 (Not used in PSoC™ 4 HV PA)
Bit [31]	Flash Controller Number	0: Controller 0 (which has Macro 0 and 1) 1: Controller 1 (which has Macro 0 only) (Refer to the **“Flash Memory”** on page 67 for the flash controllers in the device) **Note:** Each flash controller starts from row 0.
SRAM Address- 32’hYY + 0x04		
Bits [7:0]	Load Size	Number of bytes to be written to the page latch buffer. Supported values: 0x07 - 8 byte …. 0x7F - 128 bytes
Bits [15:8]	0xXX	Don’t care parameter
Bits [23:16]	0xXX	Don’t care parameter
Bits [31:24]	0xXX	Don’t care parameter
SRAM Address - From (32’hYY + 0x08) to (32’hYY + 0x08 + Load Size)		
Byte 0	Data Byte [0]	First data byte to be loaded
….	….	….
Byte (Load size –1)	Data Byte [Load size –1]	Last data byte to be loaded

Table 25-4. Parameters (continued)

Address	**Value to be Written**	**Description**
CPUSS_SYSARG register		
Bits [31:0]	32’hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1).
CPUSS_SYSREQ register		
Bits [15:0]	0x0004	Load Flash Bytes opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Table 25-5. Return

Address	**Return Value**	**Description**
CPUSS_SYSARG register		
Bits [31:0]	0xAXXXXXXX (X = don’t care)	Success status code
Bits [31:0]	0xF00000YY (Y = failure code)	Failure status code (See **“System Call Status”** on page 399)

25.5.3 Write Row

This function erases and then programs the addressed row of flash with the data in the page latch buffer. If all data in the page latch buffer is 0, then the program is skipped. The parameters for this function are stored in SRAM. The start address of the stored parameters is written to the CPUSS_SYSARG register. This function clears the page latch buffer contents after the row is programmed.

Usage Requirements: Clocks must be configured correctly as documented in **"Configure Clock"** on page 394. Call the Load Flash Bytes function before calling this function. This function can do a write operation only if the corresponding flash row is not write-protected.

Refer to the CLK_IMO_CONFIG register in the *PSoC™ 4 HV PA Registers TRM* for more information.

Parameters

Table 25-6. Parameters

Address	Value to be Written	Description
SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD8	Key2
Bits [30:16]	Row ID	Row number to write 0x0000 - Row 0
Bit [31]	Flash Controller Number	0: Controller 0 1: Controller 1 (Refer to the **"Flash Memory"** on page 67 for the flash controllers in the device) **Note:** Each flash controller starts from row 0.
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0005	Write Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Table 25-7. Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:0]	0xAXXXXXXX (X = don't care)	Success status code
Bits [31:0]	0xF00000YY (Y = failure code)	Failure status code (See **"System Call Status"** on page 399)

25.5.4 Program Row

This function programs the addressed row of the flash with data in the page latch buffer. If all data in the page latch buffer is 0, then the program is skipped. It clears the page latch buffer contents after the row is programmed. Note that before any program operation can be performed, the page(s) must first be erased to bring all memory cells that will be programmed to "0" state.

Usage Requirements: Clocks must be configured correctly as documented in **"Configure Clock"** on page 394. Call the Load Flash Bytes function before calling this function. The row must be in an erased state before calling this function. This function can do a program operation only if the corresponding flash row is not write-protected.

Parameters

Table 25-8. Parameters

Address	Value to be Written	Description
SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD9	Key2
Bits [30:16]	Row ID	Row number to program 0x0000 - Row 0
Bit [31]	Flash Controller Number	0: Controller 0 1: Controller 1 (Refer to the **"Flash Memory"** on page 67 for the flash controllers in the device) **Note:** Each flash controller starts from row 0.
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0006	Program Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Table 25-9. Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:0]	0xAXXXXXXX (X = don't care)	Success status code
Bits [31:0]	0xF00000YY (Y = failure code)	Failure status code (See **"System Call Status"** on page 399)

25.5.5 Non-Blocking Write Row

This function is used when a flash row needs to be written by the CM0+ CPU in a non-blocking manner, so that the CPU can continue to execute application code. This function should only be used to program data flash. If it is necessary to use non-blocking write row to program code flash, the CM0+ must execute code from SRAM or user SROM. **"Blocking and Non-Blocking System Calls"** on page 375 discusses about non-blocking system calls in detail.

This function has three phases: Preprogram, Erase, Program. Preprogram is the step in which all of the bits in the flash row are written a '1' in preparation for an erase operation. The erase operation clears all bits in the row, and the program operation writes the new data to the row.

While each phase is being executed, the CPU can execute code. When the non-blocking write row system call is initiated, the user cannot call any system call function other than the Resume Non-Blocking function, which is required for completion of the non-blocking write operation. After completion of each phase, the SPC triggers its interrupt. In this interrupt, call the Resume Non-Blocking system call.

Note: The device firmware must not attempt to put the device to sleep during a non-blocking write row. This action will reset the page latch buffer and the flash will be written with all zeros. The non-blocking operation does not return success status (0xAXXXXXXX) until the last Resume API is complete. The CPUSS.SYSARG register will reflect SRAM address during an ongoing non-blocking operation.

Usage Requirements: Clocks must be configured correctly as documented in **"Configure Clock"** on page 394. Call the Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. This is because the CM0+ CPU cannot execute code from code flash while doing the code flash erase program operations. If this function is called from the code flash memory, the result is undefined, and may return a bus error and trigger a hard fault when the code flash fetch operation is being done.

Parameters

Table 25-10. Parameters

Address	Value to be Written	Description
SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDA	Key2
Bits [30:16]	Row ID	Row number to write 0x0000 - Row 0
Bit [31]	Flash Controller Number	0: Controller 0 1: Controller 1 (Refer to the **"Flash Memory"** on page 67 for the flash controllers in the device)
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0007	Non-Blocking Write Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Table 25-11. Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:00]	This will contain the SRAM address if API execution is successful.	Success status code
Bits [31:0]	0xF00000YY (Y = failure code)	Failure status code (See **"System Call Status"** on page 399)

25.5.6 Non-Blocking Program Row

This function is used when a flash row needs to be programmed by the CM0+ CPU in a non-blocking manner, so that the CPU can continue to execute application code. This function should only be used to program data flash. If it is necessary to use non-blocking write row to program code flash, the CM0+ must execute code from SRAM or user SROM. **"Blocking and Non-Blocking System Calls"** on page 375 discusses about non-blocking system calls in detail.

While each phase is being executed, the CPU can execute code. When the non-blocking program row system call is called, the user cannot call any other system call function other than the Resume Non-Blocking function, which is required for the completion of the non-blocking write operation.

Unlike the Non-Blocking Write Row system call, this system call only has a single phase. Therefore, the Resume Non-Blocking function only needs to be called once from the SPC interrupt when using the Non-Blocking Program Row system call.

Note1: FLASH should be in erase state before calling non-blocking program row operation.

Note2: The device firmware must not attempt to put the device to sleep during a non-blocking program row. This action will reset the page latch buffer and the flash will be written with all zeros.

Usage Requirements: Clocks must be configured correctly as documented in **"Configure Clock"** on page 394. Call the Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. This is because the CM0+ CPU cannot execute code from code flash while doing code flash program operations. If this function is called from code flash memory, the result is undefined, and may return a bus error and trigger a hard fault when the code flash fetch operation is being done.

Parameters

Table 25-12. Parameters

Address	Value to be Written	Description
SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDB	Key2
Bits [30:16]	Row ID	Row number to write 0x0000 - Row 0
Bit [31]	Flash Controller Number	0: Controller 0 1: Controller 1 (Refer to the **"Flash Memory"** on page 67 for the flash controllers in the device) **Note:** Each flash controller starts from row 0.
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0008	Non-Blocking Program Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Table 25-13. Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:00]	This will contain the SRAM address if API execution is successful.	Success status code
Bits [31:0]	0xF00000YY (Y = failure code)	Failure status code (See **"System Call Status"** on page 399)

25.5.7 Resume Non-Blocking

This function completes the additional phases of erase and program that were started using the Non-Blocking Write Row and Non-Blocking Program Row system calls. This function must be called thrice following a call to Non-Blocking Write Row or once following a call to Non-Blocking Program Row from the SPC ISR. This function must be called within 25 ms. No other system calls can execute until all phases of the program or erase operation are complete. More details on the procedure of using the non-blocking functions are explained in **"Blocking and Non-Blocking System Calls"** on page 375.

Parameters

Table 25-14. Parameters

Address	Value to be Written	Description
SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDC	Key2
Bit [16]	Flash Controller Number	0: Controller 0 1: Controller 1 (Refer to the **"Flash Memory"** on page 67 for the flash controllers in the device) **Note:** Each flash controller starts from row 0.
Bits [31:15]	0xXXXX	Don't care. Not used by SROM
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0009	Resume Non-Blocking opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Table 25-15. Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:0]	This will contain the SRAM address if API execution is successful. It will be updated to 0xA0000000 only for Resume API called to complete the last operation.	Success status code
Bits [31:0]	0xF00000YY (Y = failure code)	Failure status code (See **"System Call Status"** on page 399)

25.5.8 Erase All

This function erases all user code in the FLASH main and work arrays, chip protection, and the all Flash Row Level Write Protection data in SFlash. To perform an Erase All in protected mode, the user must call the Write Protection API, to change the chip-level protection from protected to open mode.

Usage Requirements: Clocks must be configured correctly as documented in **"Configure Clock"** on page 394. This system call can be called only from the DAP in the programming mode and only if the chip protection mode is OPEN. If the chip protection mode is PROTECTED, then the Write Protection API must be used by the DAP to change the protection settings to OPEN. Changing the protection setting from PROTECTED to OPEN automatically does an erase all operation. Note that aborting the Erase All API during update of the chip protection can corrupt the entire SFlash row and may cause the device to not function.

Parameters

Table 25-16. Parameters

Address	Value to be Written	Description
SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDD	Key2
Bits [31:16]	0xXXXX	Don't care
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x000A	Erase All opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Table 25-17. Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:0]	0xAXXXXXXX (X = don't care)	Success status code
Bits [31:0]	0xF00000YY (Y = failure code)	Failure status code (See **"System Call Status"** on page 399)

25.5.9 Checksum

This function reads either the whole FLASH or a row of FLASH, and returns the sum of each byte read. When performing a checksum on the entire array, the user and privileged regions are included. When performing a checksum on a row of FLASH, the user, privileged, or SFLASH are valid rows.

Notes:

ECC is enabled when executing Checksum which might lead to unexpected results. See the following for details.

- Single bit errors are corrected

  A fault (cpuss.fault_flashc_c_ecc or cpuss.fault_flashc1_c_ecc) will be reported and is the primary method of detecting a bit error
- Two or more bit errors which generate NC error

  If CPUSS_FLASH*_CTL.FLASH_ERR_SILENT is '0', a bus error will occur and the CPU will LOCKUP. A reset is required to recover. Reset could be from the XRES pin, the Fault system (if configured), or a Watch Dog Timer.

  If CPUSS_FLASH*_CTL.FLASH_ERR_SILENT is '1', a zero value will be returned for the read, and included in the checksum. Unless the data was expected to be zero, this would likely result in a checksum error.
- Three or more bit errors which do not generate NC error (rare but possible)

  Checksum is calculated normally, and would likely result in a checksum error.

  The Checksum API behavior is intentional. If we disable ECC checking, then the ECC parity bits would not be checked.

Parameters

Table 25-18. Parameters

Address	Value to be Written	Description
CPUSS_SYSARG register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDE	Key2
Bits [28:16]	Row ID	Indicates the single row to checksum. Ignored if the Whole Flash Select bit is set. Valid values are 0 - (Number of Rows – 1) If SFLASH Select bit is set, valid values are 0 - (Number of SFlash Rows – 1)
Bit [29]	Flash Controller Number	0: Controller 0 1: Controller 1 **Note:** Ignored if Whole Flash Select is set.
Bit [30]	SFLASH Select	If set, performs checksum on the SFlash row indicated by Row ID. Ignored if Whole Flash Select is set.
Bit [31]	Whole Flash Select	Performs checksum on whole Flash (main + work). Ignores Row ID, SFlash Select and Flash Controller Number if set.
CPUSS_SYSREQ register		
Bits [15:0]	0x000B	Checksum opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Table 25-19. Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:0]	0xAXXXXXXX	Success status code X = Checksum. Calculated value is truncated to fit into 28 bits.
Bits [31:0]	0xF00000YY (Y = failure code)	Failure status code (See **"System Call Status"** on page 399)

25.5.10 Write Protection

This function programs both the flash row-level protection settings and the device protection settings in the supervisory flash row. Use this function to transition the part from one protection setting to another. The following shows the allowable protection state transitions. This also shows which transitions erase flash.

- OPEN -> PROTECTED: Sets chip protection to PROTECTED and programs flash row-level protection data.
- PROTECTED -> OPEN: Erases all User Region flash. Sets chip protection to OPEN and sets flash row-level protection to unprotected.
- OPEN -> KILL: Sets chip protection to KILL and programs flash row-level protection data (irreversible).

When the device is in KILL mode, it is not possible to change the chip protection mode to any other mode. Row-level protection does not prevent reading a user row; it just prevents it from being programmed.

The flash row-level protection settings are programmed separately for each flash macro in the device. Each row has a single protection bit. The total number of protection bytes is the number of flash rows divided by eight. The chip-level protection settings (1-byte) are stored in flash macro zero in the last byte location in row zero of the supervisory flash. The size of the supervisory flash row is the same as the user code flash row size.

The chip-level protection mode does not take effect until the device is reset as it is copied into the protection register during boot. The FLASH row protection takes effect immediately. Both can be verified by reading back the SFlash data.

Usage Requirements: Clocks must be configured correctly as documented in **"Configure Clock"** on page 394. The Load Flash Bytes function is used to load the flash protection bytes of a flash macro into the page latch buffer corresponding to the macro. The starting address parameter for the load function should be zero. The flash macro number should be one that needs to be programmed; the number of bytes to load is the number of flash protection bytes in that macro.

Then, the Write Protection function is called, which programs the flash protection bytes from the page latch to be the corresponding flash macro's supervisory row. In flash macro zero, which also stores the device protection settings, the device-level protection setting is passed as a parameter in the CPUSS_SYSARG register.

Parameters

Table 25-20. Parameters

Address	Value to be Written	Description
CPUSS_SYSARG register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xE0	Key2
Bits [23:16]	Device Protection Byte	0x01 – OPEN mode 0x02 – PROTECTED mode 0x04 – KILL mode
Bits [30:24]	Flash Macro Select	0x00 – Flash Macro 0 0x01 – Flash Macro 1 0x02 – Flash Macro 2 (Not used in PSoC™ 4 HV PA) 0x03 – Flash Macro 3 (Not used in PSoC™ 4 HV PA)
Bit [31]	Flash Controller Number	0: Controller 0 (which has Macro 0 and 1) 1: Controller 1 (which has Macro 0 only) (Refer to the **"Flash Memory"** on page 67 for the flash controllers in the device) **Note:** Each flash controller starts from row 0.

Table 25-20. Parameters (continued)

Address	Value to be Written	Description
CPUSS_SYSREQ register		
Bits [15:0]	0x000D	Write Protection opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Table 25-21. Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:0]	0xAX000000 (X = don't care)	Success status code
Bits [31:0]	0xF00000YY (Y = failure code)	Failure status code (See **"System Call Status"** on page 399)

25.5.11 Configure Clock

This function initializes the clock necessary for code flash programming and erasing operations. Calling this API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set appropriately before calling the flash write, program, or erase APIs. The flash write and erase APIs perform frequency checks and will exit without acting on the flash and return the "Invalid Flash Clock" status clocks that are not configured correctly.

When this API is not called before calling a flash operation, the user must ensure that the clock configuration meets the following requirements (See the **"Clocking system"** on page 93 for more details):

- When IMO clock is used:
  - CLK_IMO_SELECT register is set to 48 MHz or 49.152 MHz
  - PUMP_SEL register is set to IMO
- When EXTCLK / ECO clock is used:
  - HFCLK_SEL register is set to EXTCLK or ECO
  - HFCLK_DIV register is set to 0x0: NO_DIV
  - PUMP_SEL register is set to HFCLK

Note: Flash programming mandates the pump clock to be running at 44 MHz to 52 MHz.

Parameters

Table 25-22. Parameters

Address	Value to be Written	Description
CPUSS_SYSARG Register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xE8	Key2
Bits [31:16]	0x0000	Not used
CPUSS_SYSREQ register		
Bits [15:0]	0x0015	Command set IMO 48 MHz
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Table 25-23. Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:0]	0xAXXXXXXX (X = don't care)	Success status code
Bits [31:0]	0xF00000YY (Y = failure code)	Failure status code (See **"System Call Status"** on page 399)

25.5.12 Write User SFlash Row

This API writes a user row in supervisory flash (SFlash). PSoC™ 4 HV PA has 1KB SFlash for application-specific use. The application can store any information here. Each application should determine whether it needs this flash region and for what purpose. Also, user SFlash rows are not stored in the hex file. A vendor should define the programming process - during production, where to get the SFlash data from, and at which row/address to store it. This API does not write rows containing security or protection settings. If the row provided is out of the acceptable range, the API will return the Invalid Address status code.

SFlash user rows belong to macro 1 of Flash Controller 0. It must be specified while loading data to be programmed by the Load Flash Bytes API.

Parameters

Table 25-24. Parameters

Address	Value to be Written	Description
SRAM Address - 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xEB	Key2
Bits [31:16]	0xXXXX	Don't care parameter
SRAM Address - 32'hYY + 0x04		
Bits [7:0]	User SFlash Row	Row number to write 0x00 – User SFlash Row 0
Bits [31:8]	0xXXXX	Don't care parameter
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0018	Write User SFlash Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Table 25-25. Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:0]	0xA0000000	Success status code
Bits [31:0]	0xF00000YY (Y = failure code)	Failure status code (See **"System Call Status"** on page 399)

25.5.13 Soft Reset

This function issues a software reset by setting the CM0P_AIRCR.SYSRESETREQ bit. This API provides easy reset functionality to parts that lack a reset pin.

Parameters

Table 25-26. Parameters

Address	Value to be Written	Description
CPUSS_SYSARG Register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xEE	Key2
Bits [31:16]	0xXXXX	Don’t care parameter
CPUSS_SYSREQ register		
Bits [15:0]	0x001B	Soft reset opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Table 25-27. Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:0]	0xAXXXXXXX (X = don’t care)	Success status code
Bits [31:0]	0xF00000YY (Y = failure code)	Failure status code (See **“System Call Status”** on page 399)

25.5.14 Bulk Erase

This function erases a single macro or all macros of regular flash of a specific flash controller in bulk mode. The API does not touch SFlash. Optionally, checksum of the erased memory is used to verify whether all data is erased. A bad checksum will result in a failure status.

Parameters

Table 25-28. Parameters

Address	Value to be Written	Description
CPUSS_SYSARG Register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xF0	Key2
Bits [18:16]	Flash Macro Select	0x00 – Flash Macro 0 0x01 – Flash Macro 1 0x02 – Flash Macro 2 (Not used in PSoC™ 4 HV PA) 0x03 – Flash Macro 3 (Not used in PSoC™ 4 HV PA) 0x07 – All macro of specific flash controller
Bit [19]	Flash Controller Number	0: Controller 0 (has Macro 0 and 1) 1: Controller 1 (has Macro 0 only) (Refer to the **"Flash Memory"** on page 67 for the flash controllers in the device) **Note:** Each flash controller starts from row 0.
Bit [20]	Checksum after erase	0 - disabled 1 - enabled
Bits [31:21]	0xXXX	Don't care parameter
CPUSS_SYSREQ register		
Bits [15:0]	0x001D	Bulk Erase opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Table 25-29. Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:0]	0xAXXXXXXX (X = don't care)	Success status code
Bits [31:0]	0xF00000YY (Y = failure code)	Failure status code (See **"System Call Status"** on page 399)

25.5.15 Specify external clock frequency

This function provides the frequency that is applied on EXTCLK/ECO[1] input. It is a user's responsibility to call this API when EXTCLK/ECO frequency changes and this clock source is selected to drive HFCLK in a flash API. Flash APIs that support EXTCLK/ECO check if the clocking preconditions are satisfied.

Parameters

Table 25-30. Parameters

Address	Value to be Written	Description
CPUSS_SYSARG Register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xF1	Key2
Bits [21:16]	Frequency	[1 to 52] MHz
Bits [31:22]	0xXXX	Don't care parameter
CPUSS_SYSREQ register		
Bits [15:0]	0x001E	Specify External Clock Frequency opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Table 25-31. Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:0]	0xAXXXXXXX (X = don't care)	Success status code
Bits [31:0]	0xF00000YY (Y = failure code)	Failure status code (See **"System Call Status"** on page 399)

1) ECO is not available as a clock for the PSoC™ 4 HV PA device. It will be available in future products.

25.6 System Call Status

At the end of every system call, a status code is written over the arguments in the CPUSS_SYSARG register. A success status is 0xAXXXXXXX, where X indicates don't care values or return data in the case of the system calls that return a value. A failure status is indicated by 0xF00000YY, where YY is the failure code.

Table 25-32. System Call Status Codes

Status Code (32-bit value in CPUSS_SYSARG register)	Description
AXXXXXXXh	Success – The "X" denotes a don't care value, which has a value of '0' returned by the SROM, unless the API returns parameters directly to the CPUSS_SYSARG register.
F0000001h	Invalid Chip Protection Mode – This API is not available during the current chip protection mode.
F0000003h	Invalid Page Latch Address – The address within the page latch buffer is either out of bounds or the size provided is too large for the page address.
F0000004h	Invalid Address – The row ID or byte address provided is outside of the available memory.
F0000005h	Row Protected – The row ID provided is a protected row.
F0000007h	Resume Completed – All non-blocking APIs have completed. The resume API cannot be called until the next non-blocking API.
F0000008h	Pending Resume – A non-blocking API was initiated and must be completed by calling the resume API, before any other APIs may be called.
F0000009h	System Call Still In Progress – A resume or non-blocking is still in progress. The SPC ISR must fire before attempting the next resume.
F000000Ah	Checksum Zero Failed – The calculated checksum was not zero.
F000000Bh	Invalid Opcode – The opcode is not a valid API opcode.
F000000Ch	Key Opcode Mismatch – The opcode provided does not match key1 and key2.
F000000Dh	Flash Macro Protected - The flash macro ID provided is a protected.
F000000Eh	Invalid Start Address – The start address is greater than the end address provided.
F0000010h	No Sector Erase - The Erase Sector operation is not supported.
F0000011h	API Not Instantiated - The SRAM API for the opcode is not instantiated.
F0000012h	Invalid Flash Clock - Flash erase/program operation is not supported using current clock setup.
F0000013h	Invalid Macro ID - the macro provided is outside of the available macros.
F0000015h	An SRSS register is Lock Protected

25.7 Stack requirements

API calls share the same stack space as the application code. **Table 25-33** shows the available stack required by each API call, inclusive of the stack used when NMI exception is entered.

Table 25-33. Stack Consumption in SROM API

Function Name	Stack Consumption
Silicon ID	128
Load Flash Bytes	168
Write Row	292
Program Row	292
Non-Blocking Write Row	292
Non-Blocking Program Row	292
Resume Non-Blocking	276
Erase All	416
Checksum	192
Write Protection	424
Write User Sflash Row	300
Soft Reset	120
Bulk Erase	308
Specify External Clock Frequency	128

25.8 Flash endurance concept

This product provides separate code flash and data flash regions. While similar in many respects, these offer distinct endurance behavior.

- Code flash: Endurance limit from the device datasheet defines the maximum write cycles performed each segment of code flash.
- Data flash: Endurance limit from the device datasheet defines the maximum number of write cycles performed on each row, provided that certain assumptions are met.

25.8.1 Flash capabilities

The details of PSoC™ 4 HV PA Flash capabilities are as follows:

- A write requires erase and program (128 bytes)
  - Erase sets all bits to '0'
  - Bits can be set to '1' during any later programming cycle. Each bit should be programmed only once between erase cycles.

Endurance refers to the number of programming cycles that the flash can tolerate before the ability of the flash to store data is degraded.

25.8.2 Code Flash details

Code flash is organized into segments of up to 64KB. The endurance limit applies to each segment. A full 64KB segment has 512 rows (128 bytes per row). The total number of times the code flash can be reprogrammed is related to the number of rows used.

For example, assume a product supports 128KB of code flash and all available code flash is used. With an endurance limit of 100K cycles, we can calculate that 100K / 512 rows = 195 times that the entire code flash can be reprogrammed.

User accessible supervisory flash, if available, is part of the code flash. It is subject to the same reprogramming limits as the main code flash.

25.8.3 Data Flash details

Data flash is intended to be used for data logging and storage of other frequently changed data. To support this use case, data flash supports a per row endurance limit provided that EEPROM emulation software is used to enforce a cyclic write pattern. Cyclic writing means sequentially writing each row, up to the data flash capacity limit.

For example, an 8KB data flash will have 64 rows. Each row should be programmed in sequence (row 0, 1, 2, ... 62, 63, 0, 1, 2, ...).

EEPROM emulation software typically splits each row into header and data fields to enable cyclic writing. It also typically uses redundancy to ensure that data is not lost in the case where a part reset occurs during a write cycle.

Revision history

Document version	Date of release	Description of changes
**	2020-08-12	Initial version of PSoC 4 HV Precision Analog TRM.
*A	2020-11-24	1. Introduction chapter - Updated Figure 1-1. (ROM:32KB, 11x GPIOs) - Updated number of GPIOs. 4. Cortex-M0+ CPU chapter Updated Table 4-1. (Address Range, Description of User SFLASH) 5. DMA Controller Modes chapter - Updated 5.5 Register List (Updated feature of DMAC_INTR, Added PERI registers) 7. Flash Memory chapter - Updated Table 7-1. (Memory Region, FLASH Macro) - Updated description for 7.4.3 ECC Error Reporting - Changed section 7.5 title to Flash Wait States - Updated Table 7-6 (Removed “MCP Cycles” and “STA Closure Requirements” columns) and removed notes - Removed CPUSS_PRIV_FLASH/CPUSS_PRIV_FLASHC1 registers 8. SRAM chapter - Removed CPUSS_PRIV_RAM register 9. Fault Subsystem chapter - Added detailed description in 9.2 Fault Reporting Assignments - Updated Table 9-1. (Added information of DATA0/1) 10. Clocking System chapter - Updated 10.2.4 Precision Internal Low-speed Oscillator - Updated Figure 10-5 - Updated 10.6 Register List (Added Lock Protected, Added PERI_DIV_CMD, PERI_PCLK_CTLx) 11. Power Supply and Monitoring chapter - Changed value of battery input resistor from 10 Ω to 15 Ω - Updated Figure 11-1 and Figure 11-2 to add battery input resistor value - Updated Table 11-1 (Corrected typo) - Corrected typo (RES_CAUSE_KEY to RES_CAUSE) - Removed PWR_DDFT_SELECT register 13. Power Modes chapter - Updated Table 13-1 (Removed Active digital regulator disable and added PILO, lifetime counter) - Updated Table 13-2. (Added HPOSC and PILO) - Added 13.5 Deep Sleep Wakeup Holdoff section - Updated 13.7 Register List (Added PWR_KEY_DELAY) 14. Watchdog Timer chapter - Updated Table 14-2 and Table 14-3 - Corrected typo of Figure 14-6 15. Reset System and Interrupts (Updated chapter name) - Added description of SRSS_INTR register - Added 15.3 SRSS Interrupts section - Updated 15.4 Register List (Added SRSS_INTR registers) and Table 15-1 title

Document version	Date of release	Description of changes
*A (contd)	2020-11-24	16. Device Security and Register Protection (Updated chapter name) - Added description of Register Protection - Changed 16.1 section name from Features to Device Security - Removed CPUSS_PROTECTION register descriptions - Added 16.2 Register protection section 17. I/O System chapter - Updated description of I/O System - Updated Figure 17-1. GPIO Block Diagram - Updated Table 17-3. PSoC 4 HV PA HSIOM Port Settings - Updated 17.7.2 / 17.7.3 / 17.7.4 - Added 17.7.5 LIN Controller (MXLIN) and 17.7.6 LIN PHY - Added Table 17-4. Alternate Pin Functions and the description - Updated description of Deep Sleep status in Table 17-5 - Updated 17.7.6 LIN PHY (Added note about LIN PHY pins) - Updated 17.8 Registers (Added GPIO_INTR_CAUSE, GPIO_DFT_IO_TEST) 19. Local Interconnect Network (LIN) chapter - Added Table 19-13. Peripheral Interconnect trigger group control registers 20. Timer, Counter, PWM chapter - Updated Table 20-3. (Added TCPWM_CNTx_TR_CTRL0 and Updated Trigger source) - Added Table 20-12. Peripheral Interconnect trigger group control registers 21. Precision Analog Channel Subsystem chapter - Updated 21.2.2 Analog DSM System - Updated 21.2.2.1 Programmable Gain Amplifier - Removed 21.2.2.5 / 21.2.2.6 / 21.2.2.7 - Updated 21.2.3 Digital Data System - Updated 21.2.3.1 Decimator/21.2.3.2 Conversion Modes/21.2.3.11 Post Processor Right Shift - Moved 21.2.4 Data Path in 21.2.2 Analog DSM System - Moved Figure 21-20 in 21.2.3 Digital Data System - Updated 21.2.2.6 Quantization Level/21.2.2.7 Data Path Multiplexing - Updated 21.2.5 Automatic Gain Control - Updated Table 21-2 Modulator cap value configuration - Updated gain value from 0.75x to 0.5x - Added Table 21-3. Typical gain, full range current range, resolution, and scalar values - Updated 21.3.2 SFLASH calibration registers - Removed "Diagnostic voltages" (Double description) in 21.1 Features - Changed Static gain range to 0.5 - 512 in 21.1 Features - Updated Figure 21-2. (Changed name of INMUX signals) - Removed Data Path (Combine other sections) in 21.2.1 PACSS Measurement and Acquisition System - Changed a paragraph in 21.2.3 to "The PACSS register configuration is divided into three sections" - Corrected some typos in 21.2.2.6 Overload Logic and 21.2.3.1 Decimator - Updated DEC_SHIFTR and RR_SHIFTR formula in 21.2.3.4 Right Shift

Document version	Date of release	Description of changes
*A (contd)	2020-11-24	- Updated 21.2.3.6 Offset & Gain Correction (added AGC =1 case, added registers name as supplement) - Added Vsample and LSB calculation in 21.2.3.8 Decimator Result Calculation - Added PACSS_DCHANx_INTR name as supplement in 21.2.3.14 Interrupts and Output Triggers - Updated description of 21.2.4 Automatic Gain Control (Gain decreased: 75%, increased: 25%, Added registers name as supplement) - Updated Table 21-2. Modulator cap value configuration (Added Digital column, added note) - Updated 21.2.3.4 Right Shift (DEC_SHIFTR, RR_SHIFTR equations) - Added Latency for data available at FIR output in 21.2.3.9 FIR Filter - Updated description of 21.2.5.4 Sequencer Timing - Updated Figure 21-27. Sequencer Timing (Changed diagnostic timing) - Updated each value of 21.2.5.4 Sequencer Timing - Added note stating "the value written to these registers will be plus 1 (E.g. Write 63 for 64)" - Added description of diagnostic timing - Added Table 21-6 DR, DR2 recommended values - Removed paragraph of "Current and Voltage measurements are..." in 21.2.5.4 Sequencer Timing to after Figure 21-27 - Added 21.2.5.5 Startup Timing Requirements and corrected typos in 21.2.5.6 Channel Chopping - Updated Figure 21-29. I/O Components (Changed name of INMUX signals) - Updated description of paragraph "The INMUX diagram..." in 21.2.6.1 Input Multiplexer. - Updated Figure 21-30. INMUX connectivity diagram - Updated Table 21-7. Positive Pin Selections (Corrected source name, Added Description, Added amuxbusA/B) - Updated Table 21-8. Negative Pin Selections (Added Description, Added amuxbusA/B) - Added Table 21-9 Pair of Positive & Negative pin selections - Updated 21.2.7.1 Channel OFFSET and GAIN calibration (Added procedure of offset scaler value) - Added Register protection section in 21.3.1 PACSS registers - Updated 21.3.1.1 PACSS_MMIO registers (Added some registers, Corrected offset) - Updated 21.3.1.2 DCHAN registers (Added some registers, Corrected offset) - Updated 21.3.1.3 ACHAN registers (Added some registers) - Updated 21.3.2 SFLASH calibration registers (Added some registers, Corrected offset) - Added 21.3.3 Peripheral Interconnect trigger group control registers - Corrected some typos 22. High-Voltage Subsystem - Added description of Output current capability for Core, GPIO, & Ext. loads - Changed VDIVIDER output signal name to vs_div0/1

Revision history

Document version	Date of release	Description of changes
*A (contd)	2020-11-24	- Removed vs_div0/1_n signals - Updated Figure 22-9. (output signals) - Updated Table 22-3. (updated signal name, removed vs_div0/1_n signals) 24. Nonvolatile Memory Programming chapter - Updated 24.2 Functional Description - Updated 24.4 Blocking and Non-Blocking System Calls - Added Write User SFLASH Row in Table 24-1 - Updated 24.5.2 Configure Clock - Updated 24.5.9 Non-Blocking Write Row and 24.5.10 Non-Blocking Program Row - Added 24.5.12 Write User SFLASH Row - Removed 24.7 Non-Blocking System Call Pseudo Code - Added 24.8 Flash Endurance Concept Changed reference document to PSoC 4 HV PA Programming Specifications - Updated Table 24-1. List of System Calls (Added OpCode column, Added Soft Reset, Updated description of Erase All, Write Protection and Write User SFLASH Row) - Arranged each section to be same as Table 24-1. - Added Failure status code description in each System Call - Updated 24.5.2 Load Flash Bytes (Description, Byte Addr, Flash Macro Select, Flash Controller Number, Load Size, Return information) - Updated 24.5.3 Write Row (Description, Flash Controller Number, Return information) - Updated 24.5.4 Program Row (Description, Flash Controller Number, Return information) - Updated 24.5.5 Non-Blocking Write Row (Description, Flash Controller Number, Return information) - Updated 24.5.6 Non-Blocking Program Row (Description, Flash Controller Number, Return information) - Updated 24.5.7 Resume Non-Blocking (Flash Controller Number, Return information) - Updated 24.5.8 Erase All (Description, Return information) - Updated 24.5.9 Checksum (Row ID, Flash Controller Number, SFLASH Select, Whole Flash Select, Return information) - Updated 24.5.10 Write Protection (Description, Flash Macro Select, Flash Controller Number, Return information) - Updated 24.5.11 Configure Clock (Return information) - Updated 24.5.12 Write User SFLASH Row (Description, Return information) - Added 24.5.13 Soft reset section - Updated Table 24-2. System Call Status Codes (Added Flash Macro Protected, No Sector Erase, Invalid Macro Id, SRSS Register Lock) Updated section block diagrams (ROM-32KB and 11x GPIOs)

Document version	Date of release	Description of changes
*B	2021-02-24	7. Flash Memory - Updated Table 7-1 (added memory size, corrected memory region) - Updated 7.4 ECC Implementation (description of ECC (Single-Bit Errors)) - Updated Table 7-6 and Table 7-7 (changed system frequency) 10. Clocking System - Updated 10.2.1.3 Flash Programming Clock (divided values and frequency range) - Updated 10.2.4.1 Trim Information (added TR_CAP description) - Updated 10.3 Clock Calibration Counters (changed from ILO to IMO/PILO, changed example values) - Corrected some typos 13. Power Modes - Updated Table 13-2 (added DMA) 14. Watchdog Timer - Removed HV and LV logic descriptions - Changed logic domain of basic WDT to V_{CCD} - Updated Table 14-2 and Table 14-3 (Bit Name: ENABLE to ENABLED) 16. Device Security and Register Protection - Updated 16.2 Register Protection (added MAGIC field name, added reference of PACSS protection) - Added 16.3 Disable SWD in firmware - Added 16.4 Privileged Registers 21. Precision Analog Channel Subsystem - Added 21.2.2.5 Dynamic Element Matching - Added 21.2.2.6 Reference system - Added 21.2.2.7 Negative Pump - Added 21.2.2.8 Chopping Configuration - Added 21.2.2.9 Power Configuration - Updated 21.2.2.8 Overload Logic (added description and note) - Updated 21.2.3.1 Decimator (added output data rate description) - Added 21.2.3.2 Circular Buffer - Updated 21.2.3.4 Left Shift (added formula of SHIFTL) - Updated 21.2.3.9 Decimator Result Calculation (added note of FIR filter registers) - Updated 21.2.5.4 Sequencer Timing (sequencer timing description) - Updated 21.2.5.6 Channel Chopping (support of channel chopping assign) - Updated 21.2.5.7 Sequencer Configuration Rules (support of channel chopping assign) - Updated 21.2.7.1 Channel OFFSET and GAIN calibration (each procedure) - Updated 21.2.7.2 Internal Temperature Calculation (added description of primary / alternate), updated formula of internal temperature calculation) 24. Nonvolatile Memory Programming - Updated Table 24-1. List of System Calls (added system call of Bulk Erase / Specify External Clock Frequency) - Updated 24.5.4 Program Row (erase procedure before program operation) - Updated 24.5.5 Non-Blocking Write Row (removed sentence of “In addition, the non-blocking...”)

Document version	Date of release	Description of changes
*B (contd)	2021-02-24	- Updated 24.5.6 Non-Blocking Program Row (erase procedure before program row operation, removed sentence of "In addition, the non-blocking...") - Updated 24.5.11 Configure Clock (description when not calling this API) - Added 24.5.14 Bulk Erase - Added 24.5.15 Specify External Clock Frequency - Updated Table 24-2. System Call Status Codes (added Status Code:F0000011h, updated description of Status Code:F0000012h)
*C	2021-05-10	All chapters - Updated number of PACSS digital channel from 3 to 4 4. Cortex-M0+ CPU - Updated Table 4-1. Cortex-M0+ Address Map (updated description of 8KB: Data region.) 5. DMA Controller Modes - Updated Table 5-1. DMA Trigger Sources (added PACSS data valid channel 3) 6. Interrupts - Updated Table 6-2. List of PSoC 4 HV PA Interrupt Sources (added IRQ23/24 and Power Mode column) 7. Flash Memory chapter - Updated Table 7-1 (updated notes of 8KB: Data region.) - Updated title of Table 7-6 and Table 7-7 10. Clocking System - Updated Figure 10-2 (added 24MHz setting and wait 5-µs, removed Figure 10-3) - Updated 10.2.1.2 Flash Programming Clock (PUMP) (corrected typo and added note about pump clock disable) - Updated 10.3 Clock Calibration Counters (updated descriptions, added equation of trim updates) - Updated Table 10-4 (removed term of "fast clock") 13. Power Modes - Updated Table 13-2. Available Peripherals (added note of Deep Sleep mode) 14. Watchdog Timer - Updated 14.4.1 Overview description - Updated 14.4.2 Firmware Usage (updated Figure 14-6 and steps recommended) 15. Reset System and Interrupts - Added 15.2 Reset Levels 17. I/O System chapter - Updated Table 17-4. Alternate Pin Functions (corrected typo of cpuss.fault_out) 19. Local Interconnect Network (LIN) - Corrected PERI_TR_GROUP3_TR_OUT_CTL x[3:0] of Table 19-1. LIN Trigger Sources 20. Timer, Counter, and PWM - Updated Table 20-2. TCPWM Trigger Sources (added PACSS data valid channel 3) 21. Precision Analog Channel Subsystem - Updated Figures 21-1, 21-2, 21-3, and 21-10 (added digital channel 4) - Updated 21.2.2.6 Reference System (added reason of VREF measurement with a gain of 1x)

Document version	Date of release	Description of changes
*C (contd)	2021-05-10	- Added 21.2.2.8 Positive Pump - Updated 21.2.3.3 Conversion Modes (added note) - Updated 21.2.3.5 Right Shift (added note) - Updated 21.2.3.6 Moving Average (added latency information) - Updated Table 21-6. Modulator Cap Value Configuration (added column of MOD, Actual MOD gain and Gain correction) - Updated Table 21-8. PACSS Trigger Sources (added PACSS data valid channel 3) - Updated Table 21-9. Trigger Multiplexer Outputs (added PACSS start conversion, digital channel #3) - Updated 21.2.5.4 Sequencer Timing (added note of DR2>0 setting) - Updated 21.2.6.2 On-die Temperature Sensor (updated formulas) - Updated 21.2.6.3 External Temperature Sensor description - Updated 21.2.7.1 Channel OFFSET and GAIN calibration (updated calibration procedure) - Updated 21.2.7.2 Internal Temperature Calculation (corrected formulas) - Updated 21.3.1.1 PACSS_MMIO (corrected offset values) 22. High-Voltage Subsystem - Updated Figure 22-4. HVREG Startup and Shutdown Timing Diagram (added timing value) 24. Nonvolatile Memory Programming - Updated 24.5.7 Resume Non-Blocking (added maximum time for this function) - Added 24.7 Stack Requirements
*D	2021-07-20	8. SRAM - Updated 8.5.2 ECC Error Injection (updated description of when ECC_INJ_EN=1) 13. Power Modes - Updated Table 13-1. PSoC 4 HV PA Power Modes (added LIN PHY wakeup source in Deep Sleep) 14. Watchdog Timer - Removed HIB_PAUSE register in Table 14-1 19. Local Interconnect Network (LIN) - Corrected typo of equation 1 to 5 - Removed Hibernate power mode in 19.6.3 Wakeup in Low-Power Mode 21. Precision Analog Channel Subsystem - Updated 21.2.2.7 Negative Pump (removed description of PACSS_ACHANx_DFT_CTL.SPARE_CTL) - Added 21.2.3.10 Offset Correction Values Alignment section - Updated 21.2.3.16 Interrupts and Output Triggers (added all PACSS interrupts description) - Updated 21.2.5.4 Sequencer Timing (added Conversion Mode Transition) - Updated 21.2.7.2 Internal Temperature Calculation (added note of offset and gain correction, added recommended current ratio measurement) 22. High-Voltage Subsystem - Updated Figure 22-6. LIN PHY Block Diagram (changed diode to PMOS) - Updated 22.2.3.5 Power Modes (added RDIV_ACT_EN description) - Updated 22.2.3.6 Truth Table (added Table 22-4)

Revision history

Document version	Date of release	Description of changes
*E	2021-12-10	7. Flash Memory chapter - Updated ECC Single-Bit Errors description (added ECC syndrome logic information) - Added 7.4.2 ECC Parity Generation by Software - Added 7.4.3 8-bit ECC Syndrome Logic 8. SRAM chapter - Added 8.5.3 ECC Parity Generation by Software - Added 8.5.4 7-bit ECC Syndrome Logic 11. Power Supply and Monitoring - Updated Figure 11-1, Figure 11-2, and Table 11-1 (changed VDDA/VCCD capacitors to 0.22-µF) - Updated 11.4.2 Brownout Detect (added description of BODVDDD, BODVCCD and BODHVSS) - Updated 11.4.3 Over-Voltage Detection (added description of OVDVDDD and OVDVCCD) - Added 11.4.4 Controls BOD and OVD section - Added 11.4.5 Monitoring Supplies with ADC section - Changed chapter number of 11.4.4 Voltage References to 11.5 13. Power Modes - Updated 13.6 Low-Power Mode Entry and Exit (added note of before executing WFI instruction) 21. Precision Analog Channel Subsystem - Updated 21.2.2.4 Delta-Sigma Modulator (added description of modulator attenuation, changed modulator gain value) - Updated 21.2.2.6 Reference System (added sentence of accuracy and SNR) - Updated 21.2.2.7 Negative Pump (added recommended clock settings) - Updated Table 21-4 (added recommended clock settings) - Updated 21.2.3.4 Left Shift (updated all description) - Updated 21.2.3.5 Right Shift (updated all description) - Updated 21.2.3.10 Offset Correction Values Alignment (updated description and Table 21-6 / 21-7) - Divided PACSS Calibration chapter to PACSS Diagnostic Features - Updated 21.2.3.15 Data Storage (added ECC parity description) - Updated 21.2.7.1 Channel OFFSET and GAIN calibration (added note of SCLR bits) - Updated 21.2.8.1 Internal Temperature Calculation (updated sentence of "The current ratio measurement ..." to "The two measurements used for ...") 22. High-Voltage Subsystem - Updated Figure 22-2 and Figure 22-3 (changed VDDA capacitor to 0.22-µF) - Updated 22.2.1 HVREG (added description of pg_hv signal routing) 23. Program and Debug Interface - Removed link of PSoC 4 HV PA Programming Specifications (typo) 24. Nonvolatile Memory Programming - Removed link of PSoC 4 HV PA Programming Specifications (typo)

Revision history

Document version	Date of release	Description of changes
*F	2022-11-09	All chapters - Updated PILO accuracy to ±5% or ±7% (based on the part number) - Updated ILO frequency to 40-kHz and removed accuracy - Updated IMO and PILO accuracy (when software calibrated) to ±1% or ±1.5% (based on the part number) 6. Interrupts - Changed IRQ5 to “Reserved” in Table 6-2 7. Flash Memory - Updated 7.1 Features (added “with ECC” in Sflash description) - Updated 7.3 Flash Controller and SPCIF (added buffers depth) 10. Clocking System - Updated 10.4 Clock Distribution (added Table 10-6 and note) - Added SFLASH_IMO_4PCT_LIM, SFLASH_IMO_3PCT_LIM and SFLASH_PILO_6PCT_LIM registers in Table 10-14 12. Chip Operational Modes - Added Boot Statuses description in 12.1 Boot 14. Watchdog Timer - Updated all descriptions of Table 14-2. CRWDT Configuration Options 15. Reset System and Interrupts - Updated 15.3 Identifying Reset Sources (added note of During initial power ramp) - Changed register name from SRSS_INTR_MASKED to SRSS_MASKED 17. I/O System - Updated 17.2.1 Digital Input Buffer (added PORT_IB_MODE_SEL description) 18. Serial Communications Block (SCB) - Updated the maximum data rate in 18.2.9 Internally and Externally Clocked SPI Operations 19. Local Interconnect Network (LIN) - Updated Figure 19-7 and Figure 19-8 - Updated the headings in sections 19.11, 19.11.2, 19.11.3, and 19.11.4. - Updated the updated description of response reception and clearing the flag in 19.11.4.2 Receive Synchronization Error - Added a new section 19.11.5 Dedicated Operation Use Case(s) 20. Timer, Counter, and PWM - Added Figure 20-5 and description of line_out/ line_compl_out polarity in 20.2.4.3 Outputs 21. Precision Analog Channel Subsystem - Updated Figure 21-2 and Figure 21-10. (removed “Wakeup”) - Updated 21.1 Features (corrected typo of “support for input voltages from -300 to +300 mV”) - Updated 21.2.2.1 Programmable Gain Amplifier (added input range for RSH/RSL pins) - Updated 21.2.2.9 Chopping Configuration (corrected CHOP_EN and HPBGR_FCHOP values) - Added 21.2.3.12 Digital Channel Filters - Updated 21.2.4 Automatic Gain Control (added SHIFTL value in Table 21-9, updated Figure 21-29, and description of modulator output)

Revision history

Document version	Date of release	Description of changes
*F (contd)	2022-11-09	- Updated 21.2.5.4 Sequencer Timing (updated sentence of “The clock cycle delay...”, added sentence of “Registers list of persisting until next start of conversion” - Added “PACSS clock cycle equations” in 21.2.5.4 Sequencer Timing - Updated 21.2.5.5 Startup Timing Requirements (added “Startup timing out of reset” and “Wakeup from deepsleep”) - Updated 21.2.7.1 Channel OFFSET and GAIN calibration (added note of “when ACHAN0 takes a diagnostic measurement ...”) - Updated 21.3.2 SFlash Calibration Registers (added SFLASH_PACSS_CHAN1_ETEMP_TRIM register) 22. High-Voltage Subsystem - Changed name of LIN control register (SL_ROUND to RF_DETECT) - Updated 22.2.2.4 Timers (added note of WAKEUP_TIMER_EN) - Updated 22.2.2.5 LIN PHY Input Pins (added description of use_alt_interface) - Updated 22.2.2.8 Power Modes (Added note in Table 22-1) - Updated 22.2.2.9 Truth Tables (added Table 22-3 use_alt_interface Truth Table) 24. Nonvolatile Memory Programming - Removed sentence of “Parameter applicable only for Flash Macro 0” in 24.5.10 Write Protection
*G	2023-04-17	Migrated to the Infineon template.
*H	2023-11-15	Replaced Cypress with Infineon in all instances across the document. Updated hyperlinks across the document. All chapters - Removed CONFIDENTIAL text 19. Serial communications block (SCB) - Updated bit number of TRIGGER_LEVEL 21. Timer, Counter, and PWM - Updated Figure 21-59 25.5.9. Checksum - Updated description Updated to new template.

Edition 2023-11-15

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email:

erratum@infineon.com

Document reference

002-29223 Rev. *H